Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра систем штучного інтелекту



**Звіт**

про виконання

**Лабораторних та практичних робіт № 6**

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

***з розділу***: « Динамічні структури. Черга. Стек. Списки. Дерева. Алгоритми обробки динамічних структур»

***Виконав:***

студент групи ШІ-11

Винницький Андрій

# **Тема роботи:**

Ознайомитись з динамічними структурами, а саме з чергою, стеком, списками та деревами, зрозуміти їхні властивості та відмінності. Крім цього написати алгоритми обробки деяких динамічних структур.

Написати коди для практичних і лабораторних робіт з ВНС, а також задач з Algotester. Залити всі файли на Github і зробити pull-request.

# **Мета роботи:**

Освоїти основні концепції черги, стеку, списків та дерев, зрозуміти їхні властивості та різницю в застосуванні. Розглянути алгоритми обробки цих динамічних структур, такі як вставка, вилучення та пошук. Написати код для розв'язання завдань з практичних та лабораторних робіт. Для задач з Algotester розробити відповідні алгоритми та реалізації для їх вирішення. Створити новий репозиторій на GitHub і завантажити всі необхідні файли, включаючи коди, відповіді на завдання та звіти.

# **Теоретичні відомості:**

1. Індивідуальний план опрацювання теорії:

* Тема №1: Theory Education Activities
  + Джерела Інформації:
    - <https://cplusplus.com/doc/oldtutorial/templates/>
    - <https://en.wikipedia.org/wiki/Dynamic_array#Growth_factor>
    - <https://www.geeksforgeeks.org/binary-tree-data-structure/>
    - <https://www.geeksforgeeks.org/types-of-binary-tree/>
    - <https://www.geeksforgeeks.org/properties-of-binary-tree/>
    - <https://www.geeksforgeeks.org/introduction-to-binary-tree-data-structure-and-algorithm-tutorials/>
    - <https://ravesli.com/urok-133-peregruzka-operatorov-vvoda-i-vyvoda/>
    - <https://ravesli.com/urok-133-peregruzka-operatorov-vvoda-i-vyvoda/>
    - <https://www.programiz.com/dsa/binary-tree>
    - <https://www.programiz.com/dsa/tree-traversal>
    - <https://www.w3schools.com/cpp/cpp_constructors.asp>
    - <https://www.algolist.net/Data_structures/Dynamic_array/Capacity_management>
    - <https://www.geeksforgeeks.org/destructors-c/>
  + Що опрацьовано:
    - Дізнався про динамічні масиви, списки та дерева та в чому їхня відмінність та їхні особливості.
    - Дізнався різні алгоритми обробки динамічних структур.
  + Статус: Ознайомлений
  + Початок опрацювання теми: 08.12
  + Звершення опрацювання теми: 10.12

# **Виконання роботи:**

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

Завдання №1: VNS Lab 10

* Варіант завдання: 9

Записи в лінійному списку містять ключове поле типу int. Сформувати

двонаправлений список. Додати в нього елемент із заданим номером,

знищити К елементів з кінця списку..

Завдання №2: Algotester Lab 5

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

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.

Завдання №3: Algotester Lab 78

* Варіант завдання: 3
* 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 value 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 - print  
  Ви не отримуєте аргументів.  
  Ви виводите усі елементи дерева через пробіл.  
  Реалізувати використовуючи перегрузку оператора <<

Завдання №4: Class Practice Task

* Деталі завдання:Звязні списки і Бінарні дерева:
* 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 ⟹ 9→7→3);

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

* Задача №4 - Віддзеркалення дерева

- 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

- 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

Завдання №5: Self Practice Task 1

* Варіант завдання 1

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 <<

Завдання №6: Self Practice Task 2

* Варіант завдання 2
* 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 рівний 2.
* **Отримання значення** i-го елементу  
  Ідентифікатор - get  
  Ви отримуєте ціле число - index, індекс елемента.  
  Ви виводите значення елемента за індексом. Реалізувати використовуючи перегрузку оператора [][]
* **Модифікація значення** i-го елементу  
  Ідентифікатор - set  
  Ви отримуєте 2 цілих числа - індекс елемента, який треба змінити, та його нове значення. Реалізувати використовуючи перегрузку оператора [][]
* **Вивід динамічного масиву на екран**  
  Ідентифікатор - print  
  Ви не отримуєте аргументів.  
  Ви виводите усі елементи динамічного масиву через пробіл.  
  Реалізувати використовуючи перегрузку оператора <<

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

Програма №1 VNS Lab 10

* Планований час на реалізацію 30 хв

Програма №2 Algotester 5v2

* Блок-схема

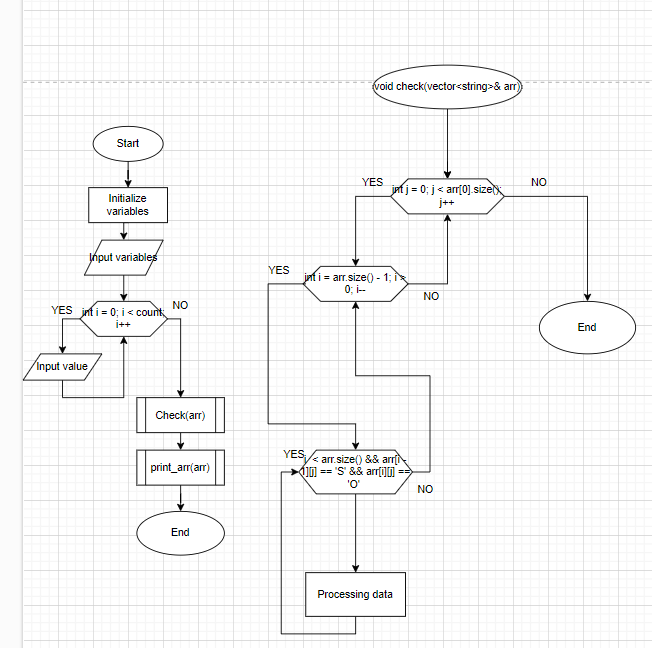
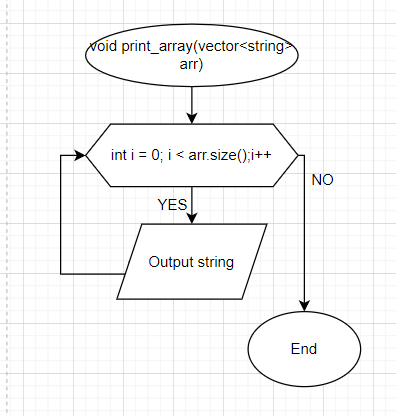
 

Figure : Блок схема до задачі 5 з Algotester

* Планований час на реалізацію 30 хв

Програма №3 Algotester 78v3

* Планований час на реалізацію 3 год.

Програма №4 Class practice

* Планований час на реалізацію 1 год.

Програма №5 Self practice 1

* Планований час на реалізацію 3 год.

Програма №6 Self practice 2

* Планований час на реалізацію 3 год.

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

Завдання №1 VNS Lab 10

Посилання: <https://github.com/artificial-intelligence-department/ai_programming_playground/pull/794/files#diff-e2e48be62b312de0007704049ff40bfad79e23b11898d9cd3f1c6a84b576cad6>

#include <iostream>  
#include <ctime>  
#include <fstream>  
  
  
template<class T>  
class TwoLinkedList  
{  
private:  
 class Node  
 {public:  
 T value;  
 Node\* next;  
 Node\* previous;  
  
  
 explicit Node(T value) {  
 this -> value = value;  
 next = nullptr;  
 previous = nullptr;  
 }  
 };  
 Node\* head;  
 Node\* tail;  
 int size;  
  
 Node\* get\_node(int index)  
 {  
 if (index < 0 || index >= size) {  
 return nullptr;  
 }  
 Node\* current;  
 int middle = size / 2;  
 if (index < middle)  
 {  
 current = head;  
 for(int j = 0; j < index;++j)  
 {  
 current = current->next;  
 }  
 }  
 else  
 {  
 current = tail;  
 for(int j = size - 1; j > index;--j)  
 {  
 current = current->previous;  
 }  
 }  
 return current;  
 }  
  
  
 void insert(Node\* new\_node, int index)  
 {  
  
 if (new\_node == nullptr || index < 0 || index > size ) {  
 return;  
 }  
  
 if (size == 0)  
 {  
 tail = head = new\_node;  
 }  
 else if(index == 0)  
 {  
 new\_node->next = head;  
 head->previous = new\_node;  
 head = new\_node;  
 }  
 else if(index == size)  
 {  
 new\_node->previous = tail;  
 tail->next = new\_node;  
 tail = new\_node;  
 }  
 else  
 {  
 Node\* current = get\_node(index - 1);  
 new\_node->previous = current;  
 new\_node->next = current->next;  
 current->next->previous = new\_node;  
 current->next = new\_node;  
 }  
  
 ++size;  
 }  
  
 void erase(int index)  
 {  
 if (index < 0 || index >= size) {  
 return; // Invalid index  
 }  
  
  
 Node\* current = get\_node(index);  
  
 if (index == 0) {  
 head = head->next;  
 if (head != nullptr) {  
 head->previous = nullptr;  
 }  
 } else if (index == size - 1) {  
 tail = tail->previous;  
 if (tail != nullptr) {  
 tail->next = nullptr;  
 }  
 } else {  
 current->previous->next = current->next;  
 current->next->previous = current->previous;  
 }  
  
 delete current;  
 --size;  
  
  
 }  
 void print(Node\* node)  
 {  
 if (node != nullptr)  
 {  
 Node\* current = node;  
 std::cout<< current->value <<' ';  
 print(current->next);  
 }  
  
 }  
  
 void write(std::string path, Node\* data)  
 {  
 std::ofstream file;  
 if(data != nullptr)  
 {  
  
 file.open(path,std::ios\_base::app);  
 if (!file){  
  
 std::cout<< "Failure" ;  
 return;  
  
 }  
 file << data->value<<"\n";  
 write(path , data->previous);  
 }  
 file.close();  
  
 }  
 void read(std::string path)  
 {  
  
 std::string line;  
 std::ifstream file;  
 file.open(path, std::ios\_base::app);  
 if (!file){  
  
 std::cout<< "Failure" ;  
  
 }  
  
 int index = 0;  
 while(getline(file , line))  
 {  
 T buff = stoi(line);  
 insert(new Node(buff),index);  
 index++;  
 }  
 file.close();  
 }  
public:  
 TwoLinkedList() : head(nullptr), tail(nullptr), size(0) {}  
  
 ~TwoLinkedList() {  
 while (head != nullptr) {  
 Node\* next = head->next;  
 delete head;  
 head = next;  
 }  
 size = 0;  
 }  
  
  
  
 void insertion(T value, int index)  
 {  
  
 insert(new Node(value) ,index);  
 }  
 void erase\_element(int index)  
 {  
 erase(index);  
 }  
 void printing()  
 {  
 print(head);  
 std::cout<<std::endl;  
 }  
 int size\_f()  
 {  
 return size;  
 }  
 void write\_to\_file(std::string path )  
 {  
 write(path, tail);  
 }void recreation(std::string path )  
 {  
 read(path);  
 }  
 T& operator[](int index){  
 return get\_node(index)->value;  
 }  
};  
  
  
int main() {  
 using namespace std;  
 TwoLinkedList<int> list;  
 srand(time(0));  
 for (int i = 0 ; i < 10; i++)  
 {  
 list.insertion(rand() % 100, i);  
 }  
  
 cout<< "Initialized list:" ;  
 list.printing();  
 int index, count;  
 cout<< "\nEnter a number of elements to delete: ";  
 cin >> count;  
 cout<< "\nEnter a index of elements before which you want delete: ";  
 cin >> index;  
 for (int j = index ; j > index - count; --j)  
 list.erase\_element(index - count - 1);  
 cout<< "\nEdited list:" ;  
 list.printing();  
 int count1;  
 cout<< "\nEnter a number of elements to add: ";  
 cin >> count1;  
 int buff = list.size\_f();  
 for (int k = list.size\_f(); k < buff + count1 ; k++)  
 {  
 list.insertion(rand() % 100, k);  
 }  
 cout<< "\nEdited list:" ;  
 list.printing();  
 list.write\_to\_file("C:/Users/Nout\_1/CLionProjects/untitled4/output.txt");  
 list.~TwoLinkedList();  
 cout<< "\nDeleted list:" ;  
 list.printing();  
 list.recreation("C:/Users/Nout\_1/CLionProjects/untitled4/output.txt");  
 cout<< "\nRecreated list:" ;  
 list.printing();  
  
  
  
  
  
  
 return 0;  
}

Завдання №2: Algotester Lab 5

Посилання: <https://github.com/artificial-intelligence-department/ai_programming_playground/pull/794/files#diff-3ce5744a168c2a516bd1f283212c18c4763748ec45c52d1d619b997e57b4cadc>

#include <iostream>  
#include <vector>  
using namespace std;  
  
void check(vector<string>& arr)  
{  
  
 for(int j = 0; j < arr[0].size(); j++)  
  
 {  
 for(int i = arr.size() - 1; i > 0; i--)  
 {  
 while(i < arr.size() && arr[i - 1][j] == 'S' && arr[i][j] == 'O' )  
 {  
 char buff = arr[i][j];  
 arr[i][j] = arr[i - 1][j];  
 arr[i - 1][j] = buff;  
 i++;  
 }  
  
  
 }  
  
 }  
 }  
  
  
  
void print\_array(vector<string> str)  
{  
 for(int i = 0; i < str.size(); i++)  
 {  
 cout << str[i]<<endl;  
 }  
}  
  
int main()  
{  
int count, count1;  
 cin>>count>> count1;  
 vector<string> arr;  
 for(int i = 0; i < count; i++)  
 {  
 string value;  
 cin>> value;  
 arr.push\_back(value);  
 }  
 check(arr);  
 print\_array(arr);

return 0;

}

Завдання №3: Algotester Lab 78

Посилання: <https://github.com/artificial-intelligence-department/ai_programming_playground/pull/794/files#diff-b64b870b0e3d5f2448d3acbc31e92ebd2ec95acb5fd6cbdf0ab9399184a8b32a>

#include <iostream>  
using namespace std;  
  
// Binary tree (search)  
  
template<class T>//custom type  
class BTS {  
 class Node {  
 public:  
 T data;  
 Node \*left;  
 Node \*right;  
 Node \*parent;  
  
 Node(T value, Node \*left\_ptr, Node \*right\_ptr, Node \*parent\_ptr) {  
 data = value;  
 left = left\_ptr;  
 right = right\_ptr;  
 parent = parent\_ptr;  
 }  
  
 // explicit prohibits this kind of shit:  
 // Node\* ptr = 15;  
  
 // Now only this is legal  
 // Node\* ptr = new Node(15);  
 // Node\* ptr(15);  
  
 // we can reuse constructors of this class and base classes  
 explicit Node(T value) : Node(value, nullptr, nullptr, nullptr) {}  
 };  
  
 int size;  
 Node \*root;  
  
 void insert\_element(Node \*head, Node \*new\_node) {  
 if (new\_node == nullptr) return;  
 if (head == nullptr) {  
 root = new\_node;  
 size++;  
 return;  
 }  
 Node \*current = head;  
  
 if (current->data > new\_node->data) {  
 if (current->left == nullptr) {  
 current->left = new\_node;  
 new\_node->parent = current;  
 new\_node->left = nullptr;  
 new\_node->right = nullptr;  
 size++;  
 return;  
 }  
 insert\_element(current->left, new\_node);  
 } else if (current->data < new\_node->data) {  
 if (current->right == nullptr) {  
 current->right = (new\_node);  
 new\_node->parent = current;  
 new\_node->left = nullptr;  
 new\_node->right = nullptr;  
 size++;  
 return;  
 }  
 insert\_element(current->right, new\_node);  
  
 }  
 }  
 void delete\_BTS(Node\* head) {  
 if (head == nullptr) return;  
  
 delete\_BTS(head->left);  
 delete\_BTS(head->right);  
 delete head;  
 }  
  
 void contain(Node \*node, T value) {  
 Node \*current = node;  
 if (current == nullptr) {  
 cout << "No";  
 return;  
 } else {  
 if (current->data == value) {  
 cout << "Yes";  
 return;  
 } else {  
 if (current->data > value) {  
 contain(current->left, value);  
 } else {  
 contain(current->right, value);  
 }  
  
 }  
 }  
 }  
  
 void remove(Node \*node) {  
 if (node->parent == nullptr) {  
 root = nullptr;  
 Node \*left = node->left;  
 Node \*right = node->right;  
 size--;  
 delete node;  
 if (left != nullptr) {  
 left->parent = nullptr;  
 root = left;  
 insert\_element(root, right);  
 } else if (right != nullptr) {  
 right->parent = nullptr;  
 root = right;  
 }  
  
 }  
 if (node->parent->left == node) {  
 node->parent->left = nullptr;  
 } else {  
 node->parent->right = nullptr;  
 }  
 Node \*left = node->left;  
 Node \*right = node->right;  
 size--;  
 delete node;  
 insert\_element(root, left);  
 insert\_element(root, right);  
  
 }  
  
 void print(Node \*node, std::ostream& out) const {  
 Node \*current = node;  
 if (current == nullptr) {  
 return;  
 }  
 print(current->left, out);  
 out << current->data << ' ';  
 print(current->right, out);  
 }  
  
public:  
 BTS() {  
 root = nullptr;  
 size = 0;  
 }  
  
 ~BTS() {  
 delete\_BTS(root);  
 }  
  
 void insertion(T value) {  
 insert\_element(root, new Node(value));  
 }  
  
 void check(T value) {  
 contain(root, value);  
 }  
  
 int count\_size() {  
 return size;  
 }  
  
 void print\_bts(std::ostream& out) const {  
 print(root, out);  
 }  
  
};  
template<class T>  
// output stream  
// cout << tree << endl;  
// operator<<(operator<<(cout, tree), endl);  
std::ostream& operator<<(std::ostream &out, const BTS<T>& tree)  
{  
 tree.print\_bts(out);  
 return out;  
}  
  
int main() {  
 BTS<int> tree;  
 int queries;  
 string query;  
 cin >> queries;  
  
 for (int i = 0; i < queries; ++i) {  
 cin >> query;  
  
  
 if (query == "insert") {  
 int element;  
 cin >> element;  
 tree.insertion(element);  
 } else if (query == "size") {  
 cout << tree.count\_size() << endl;  
 } else if (query == "print") {  
 cout << tree << endl;  
 } else if (query == "contains") {  
 int element;  
 cin >> element;  
 tree.check(element);  
 cout << endl;  
 }  
 }  
  
 return 0;  
}

Завдання №4: Class Practice Task part 1

Посилання: <https://github.com/artificial-intelligence-department/ai_programming_playground/pull/794/files#diff-bf43b83c59118a5d80beff8a764f839ab8aefacd19d473b24e554de169758cb3>

#include <iostream>  
template<class T>  
class TwoLinkedList  
{  
private:  
 class Node  
 {public:  
 T value;  
 Node\* next;  
 Node\* previous;  
  
  
 explicit Node(T value) {  
 this -> value = value;  
 next = nullptr;  
 previous = nullptr;  
 }  
 };  
 Node\* head;  
 Node\* tail;  
 int size;  
  
 Node\* get\_node(int index)  
 {  
 if (index < 0 || index >= size) {  
 return nullptr;  
 }  
 Node\* current;  
 int middle = size / 2;  
 if (index < middle)  
 {  
 current = head;  
 for(int j = 0; j < index;++j)  
 {  
 current = current->next;  
 }  
 }  
 else  
 {  
 current = tail;  
 for(int j = size - 1; j > index;--j)  
 {  
 current = current->previous;  
 }  
 }  
 return current;  
 }  
  
  
 void insert(Node\* new\_node, int index)  
 {  
  
 if (new\_node == nullptr || index < 0 || index > size ) {  
 return;  
 }  
  
 if (size == 0)  
 {  
 tail = head = new\_node;  
 }  
 else if(index == 0)  
 {  
 new\_node->next = head;  
 head->previous = new\_node;  
 head = new\_node;  
 }  
 else if(index == size)  
 {  
 new\_node->previous = tail;  
 tail->next = new\_node;  
 tail = new\_node;  
 }  
 else  
 {  
 Node\* current = get\_node(index - 1);  
 new\_node->previous = current;  
 new\_node->next = current->next;  
 current->next->previous = new\_node;  
 current->next = new\_node;  
 }  
  
 ++size;  
 }  
  
 void print(Node\* node , std::string symbol)  
 {  
 if (node != nullptr)  
 {  
 Node\* current = node;  
 std::cout<< current->value <<symbol;  
 print(current->next, symbol);  
 }  
  
 }  
  
 void reversed(Node\* node )  
 {  
 Node\* current = node;  
 while(current != nullptr)  
 {  
 Node\* buff = current->next;  
 current->next = current->previous;  
 current->previous = buff;  
 current = buff;  
 }  
 Node\* buff = head;  
 head = tail;  
 tail = buff;  
  
 }  
  
public:  
 TwoLinkedList() : head(nullptr), tail(nullptr), size(0) {}  
  
 ~TwoLinkedList() {  
 while (head != nullptr) {  
 Node\* next = head->next;  
 delete head;  
 head = next;  
 }  
 size = 0;  
 }  
 void reverse()  
 {  
 reversed(head);  
 }  
  
 bool compare(Node\* node1, Node\* node2)  
 {  
 Node\* current1 = node1;  
 Node\* current2 = node2;  
 while(current1 != nullptr && current2 != nullptr)  
 {  
 if(current1->value != current2 -> value)  
 {  
 return false;  
 }  
 current1 = current1->next;  
 current2 = current2-> next;  
 }  
 return true;  
 }  
  
 void insertion(T value, int index)  
 {  
  
 insert(new Node(value) ,index);  
 }  
  
 void insertion(Node\* value, int index)  
 {  
  
 insert(value ,index);  
 }  
  
 void sum(Node\* node1, Node\* node2, TwoLinkedList<int>& buff)  
 {  
  
 int index = 0;  
 int carry = 0;  
 Node\* current1 = node1;  
 Node\* current2 = node2;  
 while(current1 != nullptr && current2 != nullptr)  
 {  
 if((current2->value + current1->value + carry) < 10)  
 {  
 buff.insertion(current2->value + current1->value + carry, index);  
 carry = 0;  
 }  
 else  
 {  
 buff.insertion((current2->value + current1->value + carry) - 10, index);  
 carry = 1;  
 }  
 current1 = current1->next;  
 current2 = current2->next;  
 index++;  
 }  
 if (carry)  
 {  
 buff.insertion(carry, index);  
 }  
  
 }  
  
 void printing(std::string symbol)  
 {  
 print(head, symbol);  
 std::cout<<std::endl;  
 }  
  
 Node\* operator[](int index)  
 {  
 return get\_node(index);  
 }  
};  
  
  
int main() {  
 using namespace std;  
 TwoLinkedList<int> list1, list2, list3, list4, list5;  
  
 srand(time(0));  
 for (int i = 0 ; i < 10; i++)  
 {  
 list1.insertion(rand() % 100, i);  
 }  
 for (int j = 0 ; j < 10; j++)  
 {  
 list2.insertion(rand() % 100, j);  
 }  
 for (int i = 0 ; i < 6; i++)  
 {  
 list3.insertion(rand() % 10, i);  
 }  
 for (int j = 0 ; j < 6; j++)  
 {  
 list4.insertion(rand() % 10, j);  
 }  
  
  
 cout<< "Initialized list 1:" ;  
 list1.printing(" ");  
 cout<< "\nInitialized list 2:" ;  
 list2.printing(" ");  
 list1.reverse();  
 cout<< "Reversed list:" ;  
 list1.printing(" ");  
 string check = (list1.compare(list1[0],list2[0]))?("are equal"):("aren't equal");  
 cout<<"\nCompared lists: "<< check<<endl;  
 string check1 = (list1.compare(list1[0],list1[0]))?("are equal"):("aren't equal");  
 cout<<"\nCompared lists: "<< check1<<endl;  
 cout<< "\nNumber reversed 1: " ;  
 list3.printing("");  
 cout<< "\nNumber reversed 2: " ;  
 list4.printing("");  
 list5.sum(list3[0], list4[0], list5);  
 cout<< "\nSum: ";  
 list5.reverse();  
 list5.printing("");  
 return 0;  
  
}

Завдання №4: Class Practice Task part 2

Посилання: <https://github.com/artificial-intelligence-department/ai_programming_playground/pull/794/files#diff-b5df2d62f59c54ed09fa10c3fa6358d7ef06e05bc47be5d0cfbfd81cf8bf1110>

#include <iostream>  
#include<ctime>  
using namespace std;  
  
// Binary tree (search)  
  
template<class T>//custom type  
class BTS {  
 class Node {  
 public:  
 T data;  
 Node \*left;  
 Node \*right;  
 Node \*parent;  
  
 Node(T value, Node \*left\_ptr, Node \*right\_ptr, Node \*parent\_ptr) {  
 data = value;  
 left = left\_ptr;  
 right = right\_ptr;  
 parent = parent\_ptr;  
 }  
  
 // explicit prohibits this kind of shit:  
 // Node\* ptr = 15;  
  
 // Now only this is legal  
 // Node\* ptr = new Node(15);  
 // Node\* ptr(15);  
  
 // we can reuse constructors of this class and base classes  
 explicit Node(T value) : Node(value, nullptr, nullptr, nullptr) {}  
 explicit Node(T value, Node\* parent) : Node(value, nullptr, nullptr, parent) {}  
 };  
  
 int size;  
 Node \*root;  
  
 void insert\_element(Node \*head, Node \*new\_node) {  
 if (new\_node == nullptr) return;  
 if (head == nullptr) {  
 root = new\_node;  
 size++;  
 return;  
 }  
 Node \*current = head;  
  
 if (current->data > new\_node->data) {  
 if (current->left == nullptr) {  
 current->left = new\_node;  
 new\_node->parent = current;  
 new\_node->left = nullptr;  
 new\_node->right = nullptr;  
 size++;  
 return;  
 }  
 insert\_element(current->left, new\_node);  
 } else if (current->data < new\_node->data) {  
 if (current->right == nullptr) {  
 current->right = (new\_node);  
 new\_node->parent = current;  
 new\_node->left = nullptr;  
 new\_node->right = nullptr;  
 size++;  
 return;  
 }  
 insert\_element(current->right, new\_node);  
  
 }  
 }  
 void delete\_BTS(Node\* head) {  
 if (head == nullptr) return;  
  
 delete\_BTS(head->left);  
 delete\_BTS(head->right);  
 delete head;  
 }  
  
 void contain(Node \*node, T value) {  
 Node \*current = node;  
 if (current == nullptr) {  
 cout << "No";  
 return;  
 } else {  
 if (current->data == value) {  
 cout << "Yes";  
 return;  
 } else {  
 if (current->data > value) {  
 contain(current->left, value);  
 } else {  
 contain(current->right, value);  
 }  
  
 }  
 }  
 }  
  
 void remove(Node \*node) {  
 if (node->parent == nullptr) {  
 root = nullptr;  
 Node \*left = node->left;  
 Node \*right = node->right;  
 size--;  
 delete node;  
 if (left != nullptr) {  
 left->parent = nullptr;  
 root = left;  
 insert\_element(root, right);  
 } else if (right != nullptr) {  
 right->parent = nullptr;  
 root = right;  
 }  
  
 }  
 if (node->parent->left == node) {  
 node->parent->left = nullptr;  
 } else {  
 node->parent->right = nullptr;  
 }  
 Node \*left = node->left;  
 Node \*right = node->right;  
 size--;  
 delete node;  
 insert\_element(root, left);  
 insert\_element(root, right);  
  
 }  
  
 void print(Node \*node, std::ostream& out) const {  
 Node \*current = node;  
 if (current == nullptr) {  
 return;  
 }  
 print(current->left, out);  
 out << current->data << ' ';  
 print(current->right, out);  
 }  
 Node\* mirror\_tree(Node\* current)  
 {  
 if (current == nullptr)  
 {  
 return current;  
 }  
  
 Node\* left = mirror\_tree((current->left));  
 Node\* right = mirror\_tree((current->right));  
 current->left = right;  
 current->right = left;  
  
 return current;  
 }  
 Node\* sum\_parent(Node\* current)  
 {  
 if (current == nullptr)  
 {  
 return current;  
 }  
  
 Node\* left = sum\_parent((current->left));  
 Node\* right = sum\_parent((current->right));  
 if(left != nullptr && right!= nullptr)  
 {  
 current->data = left->data + right->data;  
 }  
 else if(left == nullptr && right != nullptr)  
 {  
 current->data = right->data;  
 }  
 else if(right == nullptr && left != nullptr)  
 {  
 current->data = left->data;  
 }  
  
  
  
  
  
 return current;  
 }  
public:  
 BTS() {  
 root = nullptr;  
 size = 0;  
 }  
  
 ~BTS() {  
 delete\_BTS(root);  
 }  
 void sum()  
 {  
 sum\_parent(root);  
 }  
 void mirror()  
 {  
 mirror\_tree(root);  
 }  
 void insertion(T value) {  
 insert\_element(root, new Node(value));  
 }  
  
 void check(T value) {  
 contain(root, value);  
 }  
  
 int count\_size() {  
 return size;  
 }  
  
 void print\_bts(std::ostream& out) const {  
 print(root, out);  
 }  
  
};  
template<class T>  
// output stream  
// cout << tree << endl;  
// operator<<(operator<<(cout, tree), endl);  
std::ostream& operator<<(std::ostream &out, const BTS<T>& tree)  
{  
 tree.print\_bts(out);  
 return out;  
}  
  
int main() {  
 BTS<int> tree;  
 srand(time(0));  
 for (int i = 0 ; i < 10; i++)  
 {  
 tree.insertion(rand() % 100);  
 }  
 cout<<"Generated tree: ";  
 cout<<tree<<endl;  
 cout<<"Mirrored tree: ";  
 tree.mirror();  
 cout<<tree<<endl;  
 cout<<"Summed: ";  
 tree.sum();  
 cout<<tree<<endl;  
 return 0;  
}

Завдання №5: Self Practice Task 1

Посилання: <https://github.com/artificial-intelligence-department/ai_programming_playground/pull/794/files#diff-a63e5ce95275aee3ed968979e1f103d260877514b28820b4b534584cf78a0f89>

#include <iostream>  
template<class T>  
class TwoLinkedList  
{  
private:  
 class Node  
 {public:  
 T value;  
 Node\* next;  
 Node\* previous;  
  
  
 explicit Node(T value) {  
 this -> value = value;  
 next = nullptr;  
 previous = nullptr;  
 }  
 };  
 Node\* head;  
 Node\* tail;  
 int size;  
  
 Node\* get\_node(int index)  
 {  
 if (index < 0 || index >= size) {  
 return nullptr;  
 }  
 Node\* current;  
 int middle = size / 2;  
 if (index < middle)  
 {  
 current = head;  
 for(int j = 0; j < index;++j)  
 {  
 current = current->next;  
 }  
 }  
 else  
 {  
 current = tail;  
 for(int j = size - 1; j > index;--j)  
 {  
 current = current->previous;  
 }  
 }  
 return current;  
 }  
  
  
 void insert(Node\* new\_node, int index)  
 {  
  
 if (new\_node == nullptr || index < 0 || index > size ) {  
 return;  
 }  
  
 if (size == 0)  
 {  
 tail = head = new\_node;  
 }  
 else if(index == 0)  
 {  
 new\_node->next = head;  
 head->previous = new\_node;  
 head = new\_node;  
 }  
 else if(index == size)  
 {  
 new\_node->previous = tail;  
 tail->next = new\_node;  
 tail = new\_node;  
 }  
 else  
 {  
 Node\* current = get\_node(index - 1);  
 new\_node->previous = current;  
 new\_node->next = current->next;  
 current->next->previous = new\_node;  
 current->next = new\_node;  
 }  
  
 ++size;  
 }  
  
 void erase(int index)  
 {  
 if (index < 0 || index >= size) {  
 return; // Invalid index  
 }  
  
  
 Node\* current = get\_node(index);  
  
 if (index == 0) {  
 head = head->next;  
 if (head != nullptr) {  
 head->previous = nullptr;  
 }  
 } else if (index == size - 1) {  
 tail = tail->previous;  
 if (tail != nullptr) {  
 tail->next = nullptr;  
 }  
 } else {  
 current->previous->next = current->next;  
 current->next->previous = current->previous;  
 }  
  
 delete current;  
 --size;  
  
  
 }  
  
 void set( int index , T value)  
 {  
 Node\* modify\_node = get\_node(index);  
 modify\_node->value = value;  
 }  
 void print(Node\* node)  
 {  
 if (node != nullptr)  
 {  
 Node\* current = node;  
 std::cout<< current->value <<' ';  
 print(current->next);  
 }  
  
 }  
public:  
 TwoLinkedList() : head(nullptr), tail(nullptr), size(0) {}  
  
 ~TwoLinkedList() {  
 while (head != nullptr) {  
 Node\* next = head->next;  
 delete head;  
 head = next;  
 }  
 size = 0;  
 }  
  
 void insertion(T value, int index)  
 {  
  
 insert(new Node(value) ,index);  
 }  
 void erase\_element(int index)  
 {  
 erase(index);  
 }  
 void setter(int index, T value)  
 {  
 set(index, value);  
 }  
 void printing()  
 {  
 print(head);  
 std::cout<<std::endl;  
 }  
 int size\_f()  
 {  
 return size;  
 }  
 T& operator[](int index){  
 return get\_node(index)->value;  
 }  
};  
  
  
int main() {  
 using namespace std;  
 TwoLinkedList<int> list;  
  
 int count\_of\_questions;  
 string question;  
 cin >> count\_of\_questions;  
  
 for (int i = 0; i < count\_of\_questions; ++i) {  
 cin >> question;  
  
  
 if (question == "insert") {  
 int index, count;  
 cin >> index >> count;  
 for (int j = count - 1; j >= 0; --j)  
 {  
 int number;  
 cin >> number;  
 list.insertion(number, index);  
 index++;  
 }  
  
 } else if (question == "erase") {  
 int index, count;  
 cin >> index >> count;  
 for (int j = 0; j < count; ++j)  
 list.erase\_element(index);  
 } else if (question == "size") {  
 cout << list.size\_f() << endl;  
 } else if (question == "get") {  
 int index;  
 cin >> index;  
 cout<< list[index]<<endl;  
 } else if (question == "set") {  
 int index, value;  
 cin >> index >> value;  
 list.setter(index, value);  
 } else if (question == "print") {  
 list.printing();  
 }  
 }  
  
  
 return 0;  
}

Завдання №6: Self Practice Task 2

Посилання: <https://github.com/artificial-intelligence-department/ai_programming_playground/pull/794/files#diff-c65fef6471546a685dcd066d2f54f56ebd0e7ccc1b89982d604a18c3feddba7e>

#include<iostream>  
  
template<class T>  
class Dynamic\_arr  
{  
 const int growth\_factor = 2;  
  
 T\* data ;  
 int size;  
 int capacity;  
  
 void insert(T value, int index)  
 {  
  
 T\* buff = new T[size + 1];  
 for (int i = 0;i < size; i++) {  
 buff[i] = data[i];  
 }  
 ++size;  
 if(size >= capacity)  
 {  
 capacity \*= growth\_factor;  
 }  
 int j;  
 for (j = size - 1;j > index; j--) {  
  
 buff[j] = buff[j - 1];  
 }  
 buff[j] = value;  
 delete[] data;  
 data = buff;  
 }  
  
 void erase(int index)  
 {  
 for(int i = index; i < size - 1; i++)  
 {  
 data[i] = data[i + 1];  
 }  
 --size;  
 }  
  
  
public:  
  
 Dynamic\_arr() : data(nullptr), size(0), capacity(1) {}  
  
 T& operator[](int index)  
 {  
 return data[index];  
 }  
 void insertion(T value, int index)  
 {  
 insert(value, index);  
 }  
 void deleting(int index)  
 {  
 erase(index);  
 }  
 int size\_f()  
 {  
 return size;  
 }  
 int capacity\_f()  
 {  
 return capacity;  
 }  
 void print()  
 {  
 for(int i = 0; i < size; i++) {  
 std::cout<<data[i]<<' ';  
 }  
 }  
//destructor  
 ~Dynamic\_arr() {  
 delete[] data;  
 }  
};  
  
  
  
  
int main()  
{  
 using namespace std;  
  
 Dynamic\_arr<int> vector;  
  
 int count\_of\_questions;  
 string question;  
 cin >> count\_of\_questions;  
  
 for (int i = 0; i < count\_of\_questions; ++i) {  
 cin >> question;  
  
  
 if (question == "insert") {  
 int index, count;  
 cin >> index >> count;  
 for (int j = count - 1; j >= 0; --j)  
 {  
 int number;  
 cin >> number;  
 vector.insertion(number, index);  
 index++;  
 }  
 } else if (question == "erase") {  
 int index, count;  
 cin >> index >> count;  
 for (int j = 0; j < count; ++j)  
 vector.deleting(index);  
 } else if (question == "size") {  
 cout << vector.size\_f() << endl;}  
 else if (question == "capacity") {  
 cout << vector.capacity\_f() << endl;  
 } else if (question == "get") {  
 int index;  
 cin >> index;  
 cout<< vector[index]<<endl;  
 } else if (question == "set") {  
 int index, value;  
 cin >> index >> value;  
 vector[index] = value;  
 } else if (question == "print") {  
 vector.print();  
 cout<<endl;  
 }  
 }  
  
  
 return 0;  
}

## **4. Результати виконання завдань, тестування та фактично затрачений час:**

Завдання №1:VNS Lab 10

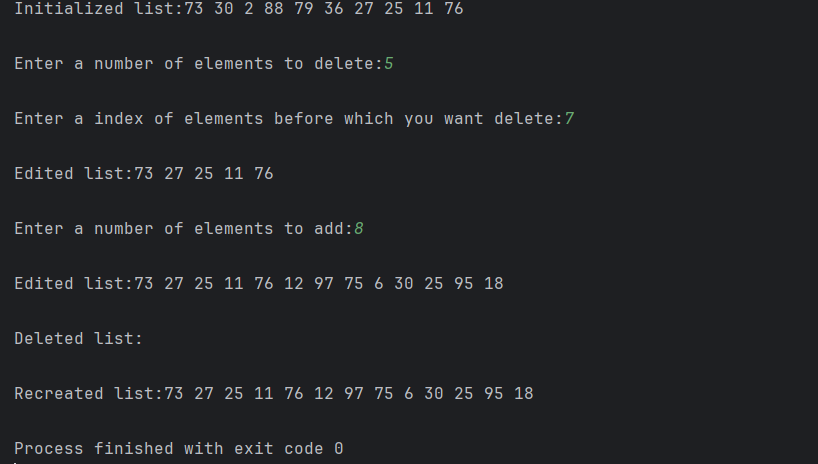
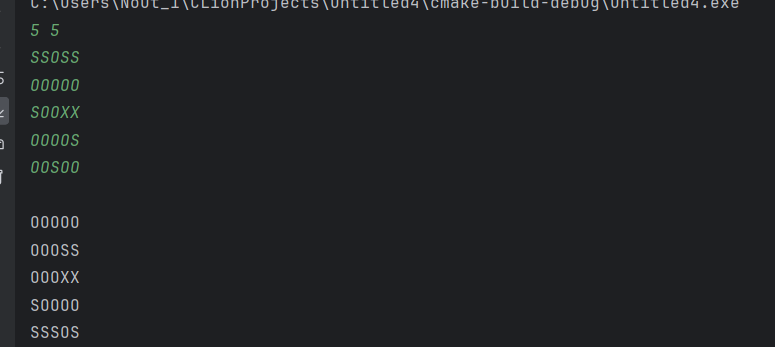


Figure :Результат виконання програми, яка створює список , видаляє , додає його елементи, записує і читає його з файлу

Час затрачений на виконання завдання: 1 год.

Завдання №2: Algotester Lab 5



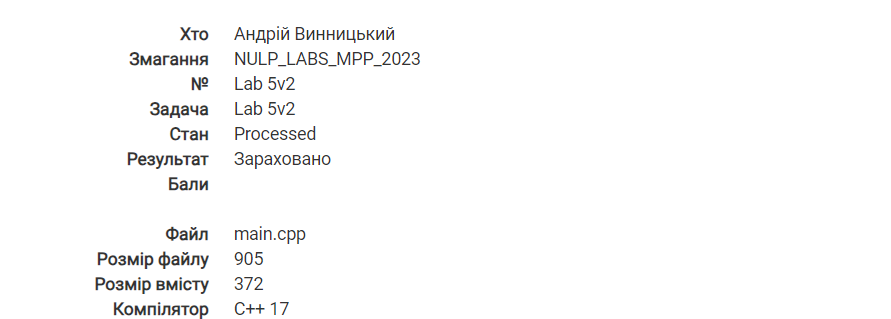
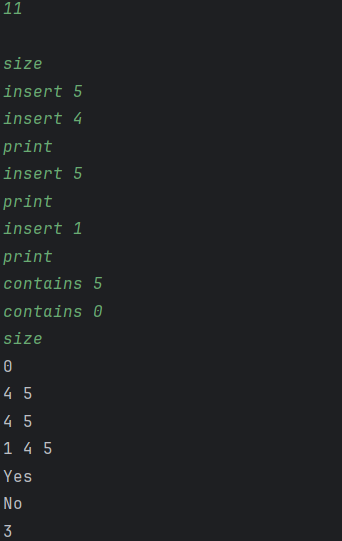


Figure : Програма виконує задачу номер 5 з алготестера

Час затрачений на виконання завдання: 30хв.

Завдання №3: Algotester Lab 78



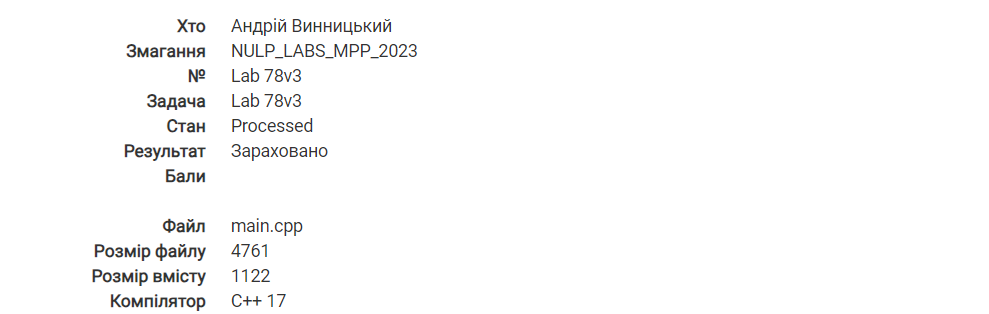


Figure : Програма створює бінарне дерево пошуку , видаляє , додає його елементи, змінює значення його елементів , виводить його розмір і саме дерево

Час затрачений на виконання завдання: 3 год

Завдання №4: Class Practice Task part1

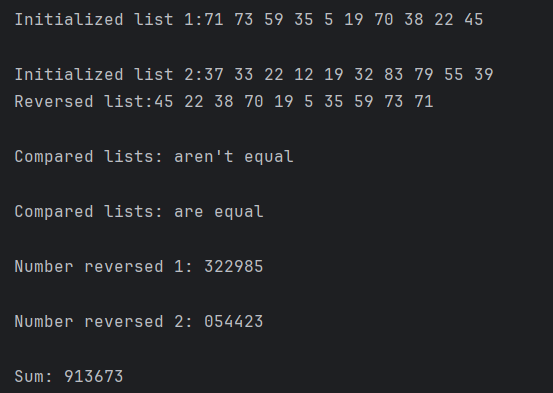


Figure :Результат виконання програми, яка створює список , робить його реверс , порівнює списки і знаходить суму великих чисел за допомогою звязних списків

Час затрачений на виконання завдання: 1.5 год

Завдання №4: Class Practice Task part2

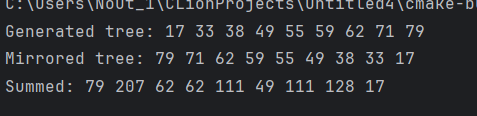
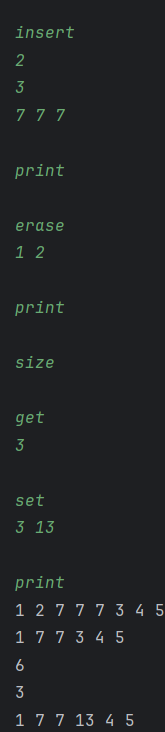


Figure :: Програма створює бінарне дерево пошуку , відзеркалює його і знаходить суму його дочірніх вузлів

Час затрачений на виконання завдання: 1.5 год

Завдання №5: Self Practice Task 1



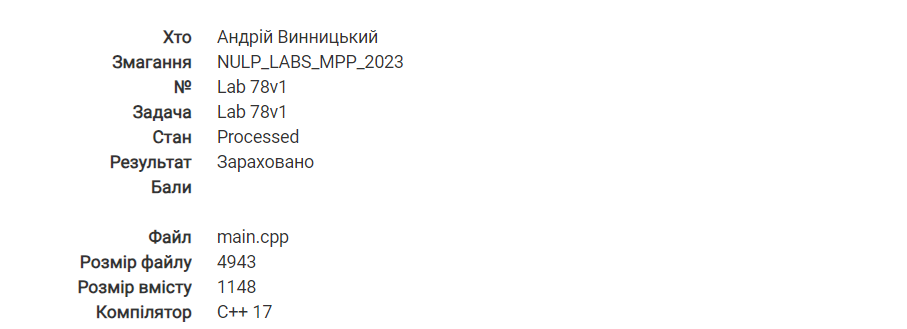
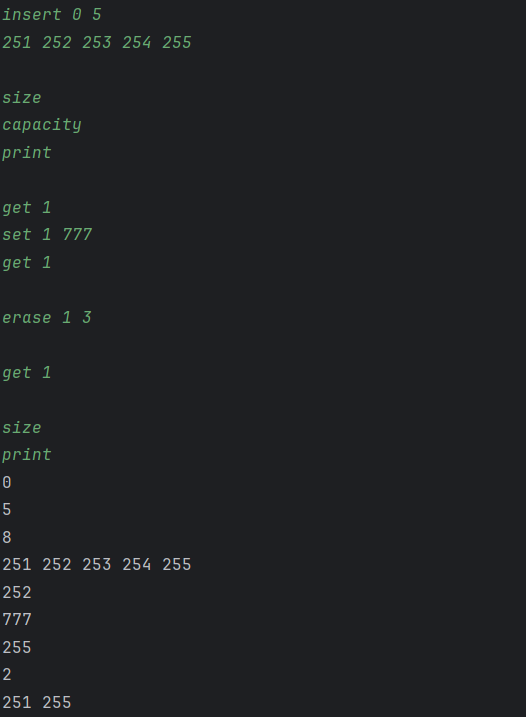


Figure : Результат виконання програми, яка створює двозв’язний список , видаляє , додає його елементи за індексом , дістає і змінює його елементи за індексом, виводить розмір і сам список

Час затрачений на виконання завдання: 3 год

Завдання №6: Self Practice Task 2



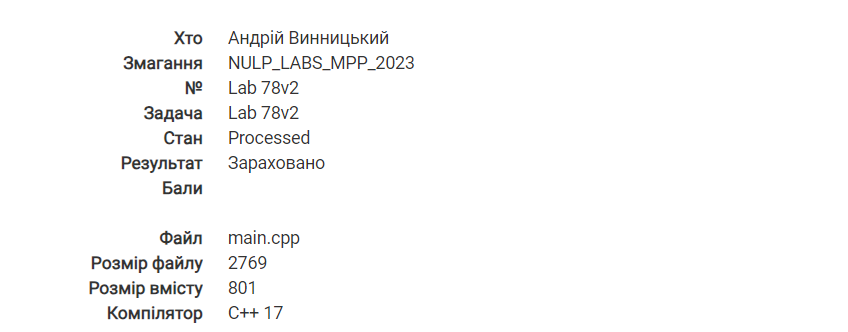


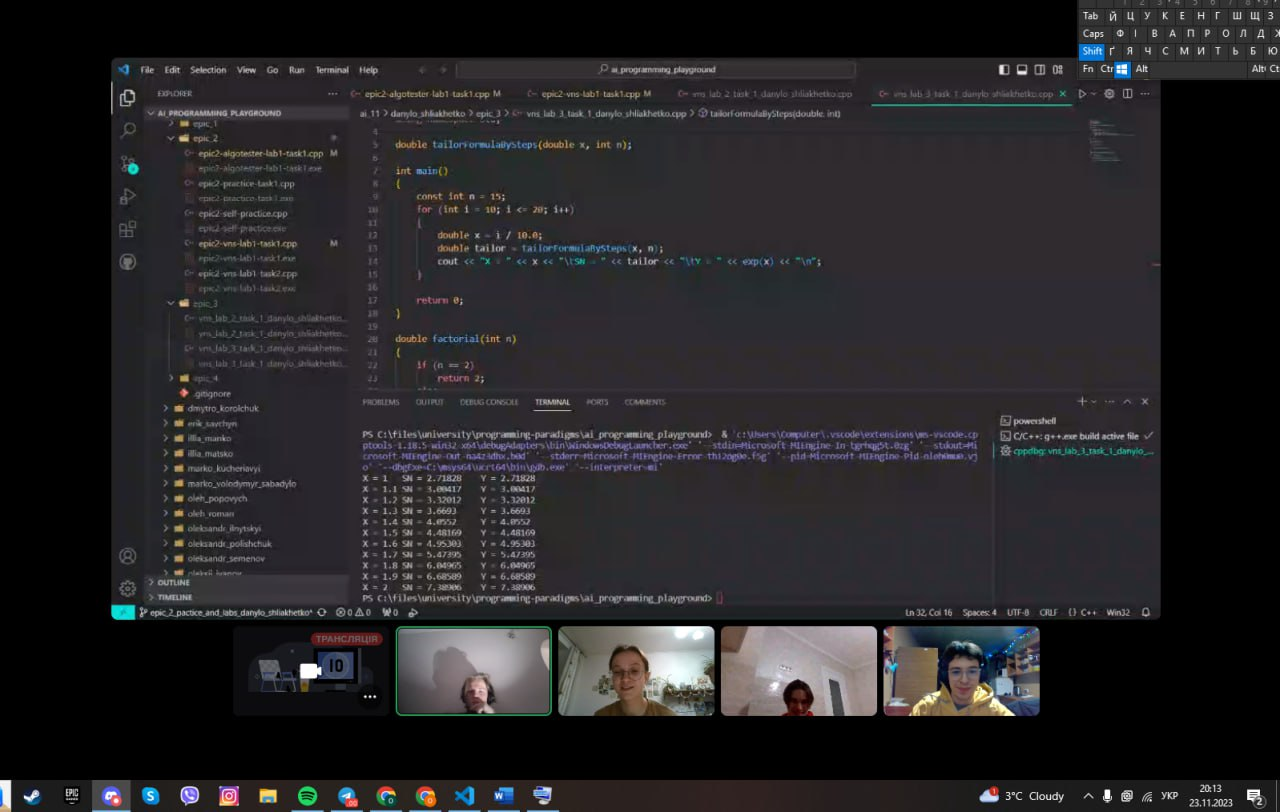
Figure : Результат виконання програми, яка створює динамічний масив , видаляє , додає його елементи за індексом , дістає і змінює його елементи за індексом, виводить розмір, зарезервовану пам’ять і сам масив

Час затрачений на виконання завдання: 2 год

## **5. Командна робота та комунікація:**

Наша команда цього тижня провела лише 1 зустріч, але дуже грунтовну:

На ній ми обговорили план на цей епік, завантажили таски у трело, а також обговорили питання стосовно коду.



*Зустріч у діскорді.*

# **Висновки:**

Під час вивчення цього епіку я освоїв новий матеріал, пов'язаний з динамічними структурами даних, такими як списки, дерева, черги та стеки. Я детально вивчив їх відмінності, ситуації, в яких вони застосовуються, а також алгоритми їх обробки. Додатково я успішно використаd отримані теоретичні знання, виконнуючи програми для ВНС та Алготестера. Всі необхідні файли, включаючи звіт, я запушив у репозиторії на GitHub.