Міністерство освіти і науки України
Фаховий коледж ракетно-космічного машинобудування
Дніпровського національного університету імені Олеся Гончара

ЗВІТ

з лабораторних робіт
з дисципліни «Алгоритми та структури даних»

Спеціальність 121
Група ПЗ-19-1

Виконав Кущевський А.П.
Перевірила Старосельцева О.В.

2022

ЗМІСТ

ЛАБОРАТОРНА РОБОТА № 1-2

Розробка основних функцій обробки зв’язаного списку. Складання та налаго-дження програм обробки однозв’язних списків.

Мета: отримання практичних навиків в формуванні та обробці динамічної структури даних – однозв’язний список.

Хід роботи

1.1 Загальна постановка задачі

1 Провести аналіз поставленої задачі.

Загальна постановка завдання:

Скласти та налагодити програму обробки однозв’язного списку за алгоритмом згідно Вашого варіанту. Програма повинна задовольняти наступним вимогам:

− організувати користувацьке меню, яке повинно містити наступні пункти:

1. Формування списку.
2. Перегляд вмісту списку.
3. Обробка списку.
4. Видалення списку.

− забезпечити коректне введення користувачем вхідних даних;

− при обробці списку враховувати, що шукані елементи можуть бути відсутні. В цьому випадку вивести користувачеві відповідне повідомлення;

− введення та виведення вхідних та вихідних даних повинно містити необхідні для користувача повідомлення.

2 Розробити та налагодити програму рішення задачі.

3 Оформити звіт з лабораторної роботи.

1.2 Постановка задачі за варіантом

9. Створити однозв’язний список дійсних чисел введенням нового елементу в початок списку. Знайти та вивести максимальний та мінімальний елементи списку та їх суму. Видалити із списку всі мінімальні елементи.

1.3 Розробка головної функції

Лістинг 1.1 – Текст програми

```
#include <iostream>
#include <Windows.h>
#include <conio.h>
#include <string>
#define UP 72
#define DOWN 80
#define ENTER 13

using namespace std;

HANDLE hStdOut = GetStdHandle(STD_OUTPUT_HANDLE);

void gotoxy(short x, short y) {

  SetConsoleCursorPosition(hStdOut, { x , y });

}

void hideCursor(bool show) {

  CONSOLE_CURSOR_INFO info;
  info.bVisible = show;
  info.dwSize = 20;
  SetConsoleCursorInfo(hStdOut, &info);

}

void clearCinBuff() {
```

```
        cin.clear();
        cin.ignore(cin.rdbuf()->in_avail());

    }

    //реализация структуры узла
    struct Node {

        float data;
        Node* ptr;//next

        Node(float value) : data(value), ptr(nullptr){}

    };

    //реализация структуры списка
    struct LinkedList {

        Node* first;
        Node* last;

        LinkedList() : first(nullptr), last(nullptr) {}

        bool is_empty() {

            return first == nullptr;

        }

        void push_begin(float value) {

            //создание нового узла, значение для поля дата передается
через конструктор
            Node* node = new Node(value);
```

```
        //если список пуст, то узел становится первым и последним
в списке
        if (is_empty()) {

            first = node;
            last = node;

        }
        else {

            node->ptr = first;
            first = node;

        }

        return;

    }

    //метод для вывода значений, которые хранятся в узлах
списка
    void printList() {

        if (is_empty()) {
            return;
        }

        //создание указателя на первый узел для прохода по всему
списку
        Node* node = first;

        //пока указатель на след. узел не nullptr
        while (node != nullptr) {

            cout << node->data << "   ";
            node = node->ptr;
```

```
        }

    }

    //метод для поиска минимального, максимального значений и
расчёта их суммы
    void minMaxSum() {

        if (is_empty()) {
            return;
        }

        Node* p = first;

        std::cout << "Список:" << std::endl << std::endl;
        printList();

        float min = p->data;
        float max = p->data;

        while (p != nullptr) {
            if (p->data < min) {
                min = p->data;
            }
            p = p->ptr;
        }

        p = first;

        while (p != nullptr) {
            if (p->data > max) {
                max = p->data;
            }
            p = p->ptr;
        }
```

```
        cout << endl;
        cout << endl << "Минимальное значение  в списке:" << min;
        cout << endl << "Максимальное значение в списке:" << max;
        cout << endl << endl << "Сумма минимального и
максимального значений:" << min + max;

        return;

    }

    //метод для удаления всех минимаьлных элементов из списка
    void deleteMin() {

        if (is_empty()) {
            return;
        }

        Node* p = first;

        float min = p->data;

        while (p != nullptr) {
            if (p->data < min) {
                min = p->data;
            }
            p = p->ptr;
        }

        Node* temp1;
        Node* temp2;
        temp1 = first;

        while (temp1 != nullptr && temp1->data == min)
        {
            first = first->ptr;
```

```
            delete temp1;
            temp1 = first;
        }
        temp2 = temp1;

        if (temp1 != nullptr)
            temp1 = temp1->ptr;

        while (temp1 != nullptr)
        {
            if (temp1->data == min)
            {
                temp2->ptr = temp1->ptr;
                delete temp1;
                temp1 = temp2->ptr;

            }
            else
            {
                temp1 = temp1->ptr;
                temp2 = temp2->ptr;
            }
        }

    }

    //метод для удаления всего списка
    void deleteList() {

        if (is_empty()) {
            return;
        }

        Node* temp = first;

        while (first != nullptr) {
```

```
            first = first->ptr;
            delete temp;
            temp = first;

        }

    }

};

LinkedList list;

void printList() {

    system("cls");
    SetConsoleTextAttribute(hStdOut, FOREGROUND_GREEN |
FOREGROUND_INTENSITY);

    if (list.is_empty()) {
        cout << "Список пуст!\n\n";
        system("pause");
        system("cls");
        return;
    }

    cout << "Список:" << endl << endl;
    list.printList();

    cout << endl << endl;
    hideCursor(false);
    system("pause");
    system("cls");
    return;

}
```

```
     void pushBegin() {

       system("cls");
       SetConsoleTextAttribute(hStdOut, FOREGROUND_GREEN |
FOREGROUND_INTENSITY);

       hideCursor(true);

       if (!(list.is_empty())) {
          cout << "На данный момент в списке присутствуют
значения!" << endl << endl;
       }

       cout << "Введите количество элементов, которые желаете
добавить в начало списка:";
       int size;
       while (!(cin >> size) || (size <= 0)) {
          clearCinBuff();
          cout << "Ошибка! Введите корректное значение:";
       }

       float data;

       cout << endl;

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

          cout << "[" << i + 1 << "]:";
          while (!(cin >> data)) {
               clearCinBuff();
               cout << "Ошибка! Введите корректное значение для ["
<< i + 1 << "]:";
          }

          list.push_begin(data);
```

```
        }

        cout << endl;
        hideCursor(false);
        system("pause");
        system("cls");
        return;

    }

    void findMinMaxSum() {

        system("cls");
        SetConsoleTextAttribute(hStdOut, FOREGROUND_GREEN |
FOREGROUND_INTENSITY);

        if (list.is_empty()) {
            cout << "Список пуст!\n\n";
            system("pause");
            system("cls");
            return;
        }

        list.minMaxSum();

        cout << endl << endl << endl;
        hideCursor(false);
        system("pause");
        system("cls");
        return;

    }

    void deleteMin() {
```

```
        system("cls");
        SetConsoleTextAttribute(hStdOut, FOREGROUND_GREEN |
FOREGROUND_INTENSITY);

        if (list.is_empty()) {
           cout << "Список пуст!\n\n";
           system("pause");
           system("cls");
           return;
        }

        string deleteMenu[] =
        {
           "Подтвердить",
           "Отклонить",
        };

        short x = 0;
        short y = 0;

        int activeMenu = 0;
        int key = 0;

        cout << "Подтвердите удаление всех минимальных элементов
списка:";

        while (true) {

           x = 0, y = 1;

           for (int i = 0; i < size(deleteMenu); ++i) {

                if (activeMenu == i) {
                     SetConsoleTextAttribute(hStdOut,
FOREGROUND_GREEN | FOREGROUND_INTENSITY);
                }
```

```
                else {
                    SetConsoleTextAttribute(hStdOut,
FOREGROUND_GREEN);
                }

                gotoxy(x, ++y);
                cout << deleteMenu[i];

            }

            key = _getch();

            switch (key) {

            case UP: {
                if (activeMenu > 0) {
                    --activeMenu;
                }
                break;
            }

            case DOWN: {
                if (activeMenu < size(deleteMenu) - 1) {
                    ++activeMenu;
                }
                break;
            }

            case ENTER: {
                switch (activeMenu) {

                case 0: {
                    SetConsoleTextAttribute(hStdOut,
FOREGROUND_GREEN | FOREGROUND_INTENSITY);
                    list.deleteMin();
                    system("cls");
```

```
                    cout << "Удаление элементов...";
                    cout << endl << endl;
                    hideCursor(false);
                    system("pause");
                    system("cls");
                    return;
               }

               case 1: {
                    SetConsoleTextAttribute(hStdOut,
FOREGROUND_GREEN | FOREGROUND_INTENSITY);
                    system("cls");
                    cout << "Возврат в главное меню...";
                    cout << endl << endl;
                    hideCursor(false);
                    system("pause");
                    system("cls");
                    return;
               }

               }
               break;
          }
          }

       }

     }

     void deleteList() {

       system("cls");
       SetConsoleTextAttribute(hStdOut, FOREGROUND_GREEN |
FOREGROUND_INTENSITY);

       if (list.is_empty()) {
```

```
            cout << "Список пуст!\n\n";
            system("pause");
            system("cls");
            return;
        }

        string deleteMenu[] =
        {
            "Подтвердить",
            "Отклонить",
        };

        short x = 0;
        short y = 0;

        int activeMenu = 0;
        int key = 0;

        cout << "Подтвердите удаление элементов из всего списка:";

        while (true) {

            x = 0, y = 1;

            for (int i = 0; i < size(deleteMenu); ++i) {

                if (activeMenu == i) {
                    SetConsoleTextAttribute(hStdOut,
FOREGROUND_GREEN | FOREGROUND_INTENSITY);
                }
                else {
                    SetConsoleTextAttribute(hStdOut,
FOREGROUND_GREEN);
                }

                gotoxy(x, ++y);
```

```
                cout << deleteMenu[i];

            }

            key = _getch();

            switch (key) {

            case UP: {
                if (activeMenu > 0) {
                    --activeMenu;
                }
                break;
            }

            case DOWN: {
                if (activeMenu < size(deleteMenu) - 1) {
                    ++activeMenu;
                }
                break;
            }

            case ENTER: {
                switch (activeMenu) {

                case 0: {
                    SetConsoleTextAttribute(hStdOut,
FOREGROUND_GREEN | FOREGROUND_INTENSITY);
                    list.deleteList();
                    system("cls");
                    cout << "Удаление элементов...";
                    cout << endl << endl;
                    hideCursor(false);
                    system("pause");
                    system("cls");
                    return;
```

```
                }

                case 1: {
                     SetConsoleTextAttribute(hStdOut,
FOREGROUND_GREEN | FOREGROUND_INTENSITY);
                     system("cls");
                     cout << "Возврат в главное меню...";
                     cout << endl << endl;
                     hideCursor(false);
                     system("pause");
                     system("cls");
                     return;
                }

                }
                break;
           }
           }

       }

     }

     void mainMenu() {

       SetConsoleTextAttribute(hStdOut, FOREGROUND_GREEN |
FOREGROUND_INTENSITY);
       hideCursor(false);

       string mainMenu[] =
       {
          "Просмотр состояния списка",
          "Добавление элементов в начало списка",
          "Поиск максимального, минимального значений и расчёт их
суммы",
          "Удаление всех минимальных значений",
```

```
            "Удаление всего списка",
            "Выход из программы"
        };

        short x = 0;
        short y = 0;

        int activeMenu = 0;
        int key = 0;

        while (true) {

            x = 10, y = 3;

            for (int i = 0; i < size(mainMenu); ++i) {

                if (activeMenu == i) {
                    SetConsoleTextAttribute(hStdOut,
FOREGROUND_GREEN | FOREGROUND_INTENSITY);
                }else{
                    SetConsoleTextAttribute(hStdOut,
FOREGROUND_GREEN);
                }

                gotoxy(x, ++y);
                cout << mainMenu[i];

            }

            key = _getch();

            switch (key) {

            case UP: {
                if (activeMenu > 0) {
                    --activeMenu;
```

```
            }
            break;
        }

        case DOWN: {
            if (activeMenu < size(mainMenu) - 1) {
                ++activeMenu;
            }
            break;
        }

        case ENTER: {
            switch (activeMenu) {

            case 0: {
                printList();
                break;
            }

            case 1: {
                pushBegin();
                break;
            }

            case 2: {
                findMinMaxSum();
                break;
            }

            case 3: {
                deleteMin();
                break;
            }

            case 4: {
                deleteList();
```

```
                break;
            }

            case 5: {
                system("cls");
                cout << "Завершение работы программы...\n\n";
                system("pause");
                exit(0);
                break;
            }

            }
            break;
        }
        }

    }

}

int main() {

    setlocale(0, "rus");
    SetConsoleTitle(L"Кущевский Андрей, ПЗ-19-1");

    mainMenu();

    _getch();
    return 0;

}
```

Вікна виконання програми дивитись на рисунках 1.1 – 1.9:

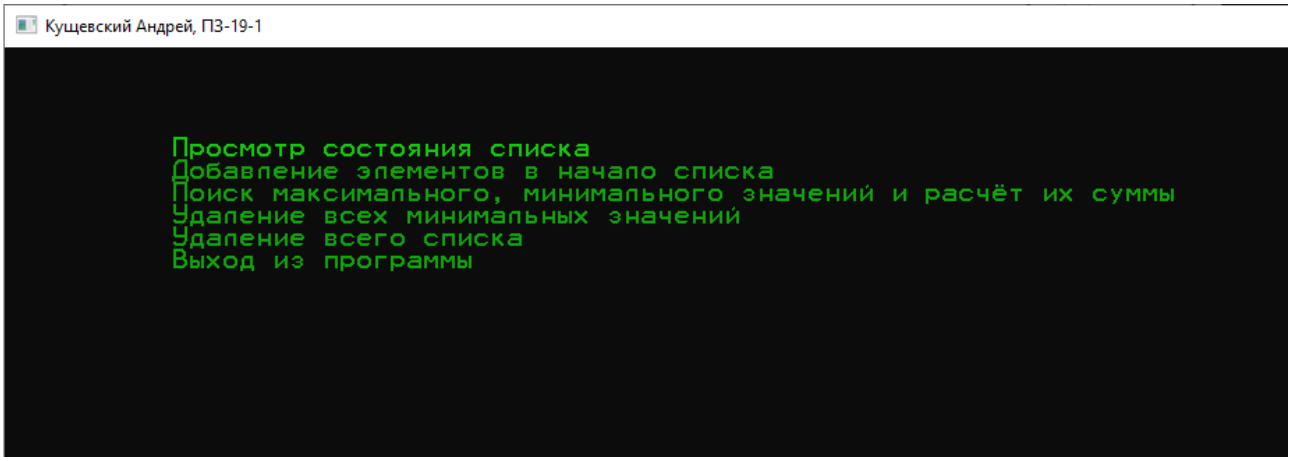


Рисунок 1.1 – Головне меню програми

Рисунок 1.2 – Перевірка користувацького друку

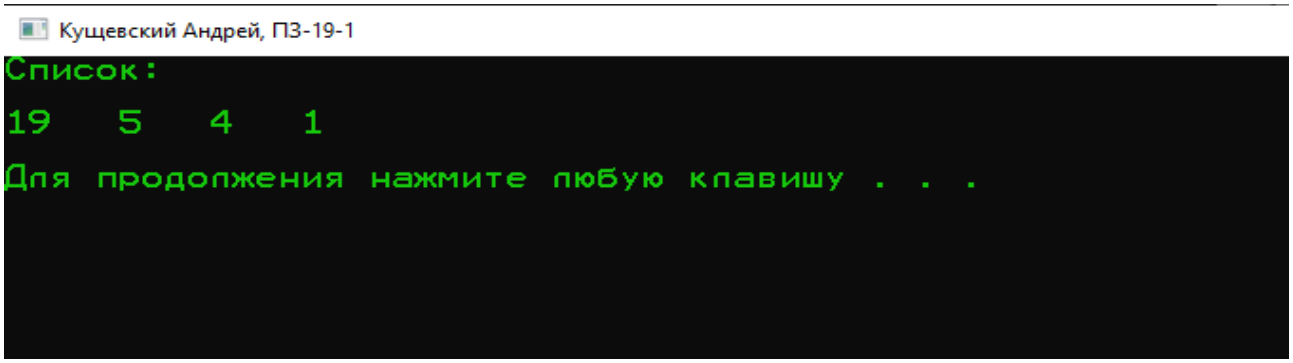


Рисунок 1.3 – Результат роботи функції showList()

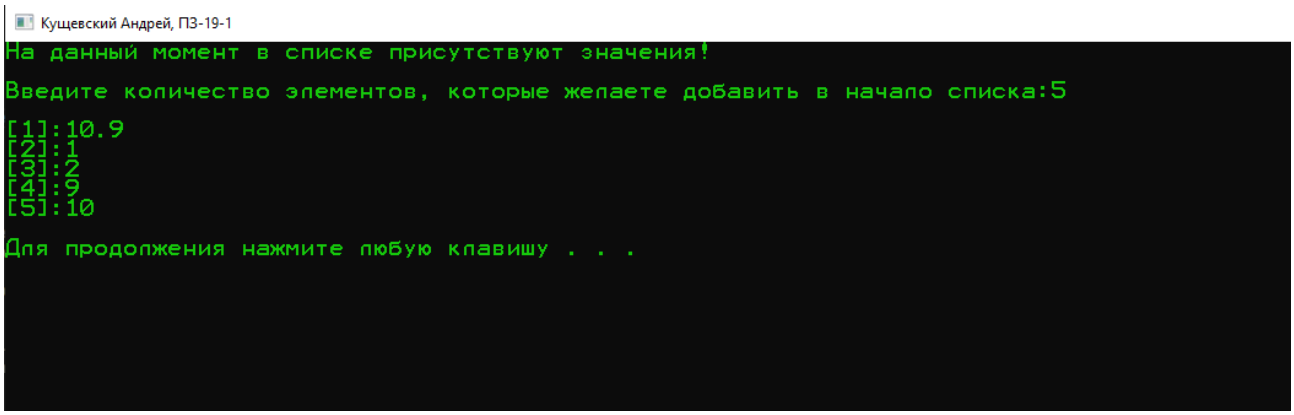


Рисунок 1.4 – Результат роботи функції createList()

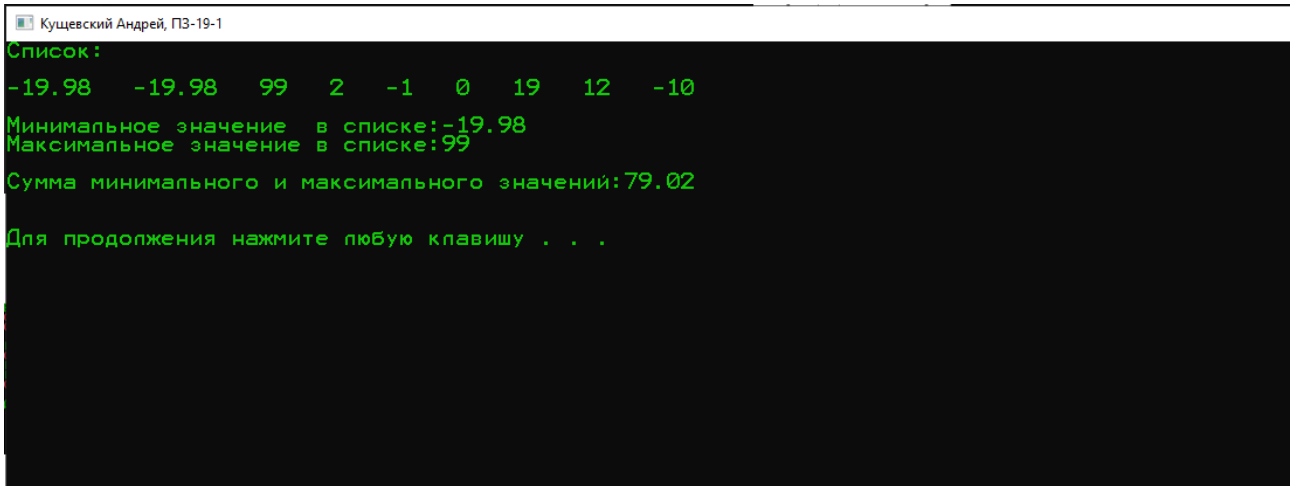
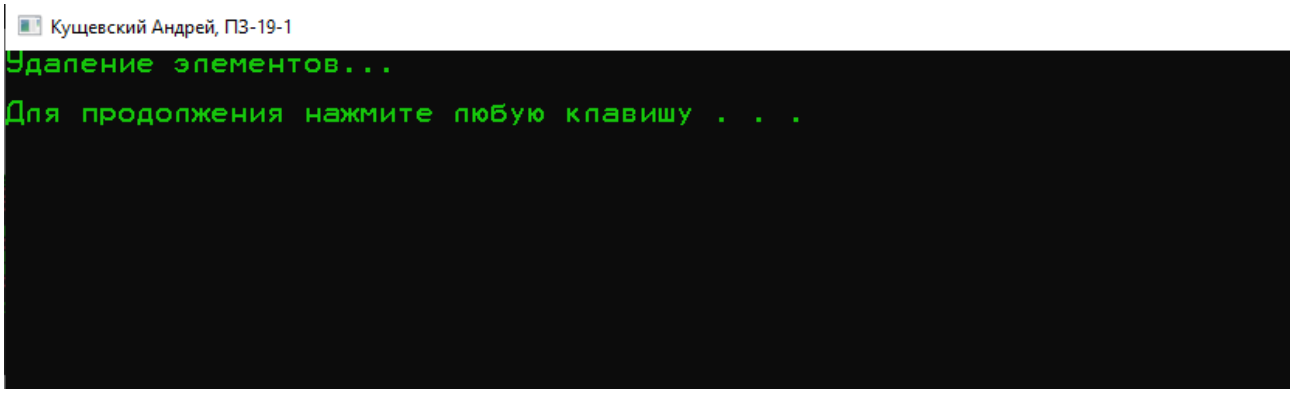
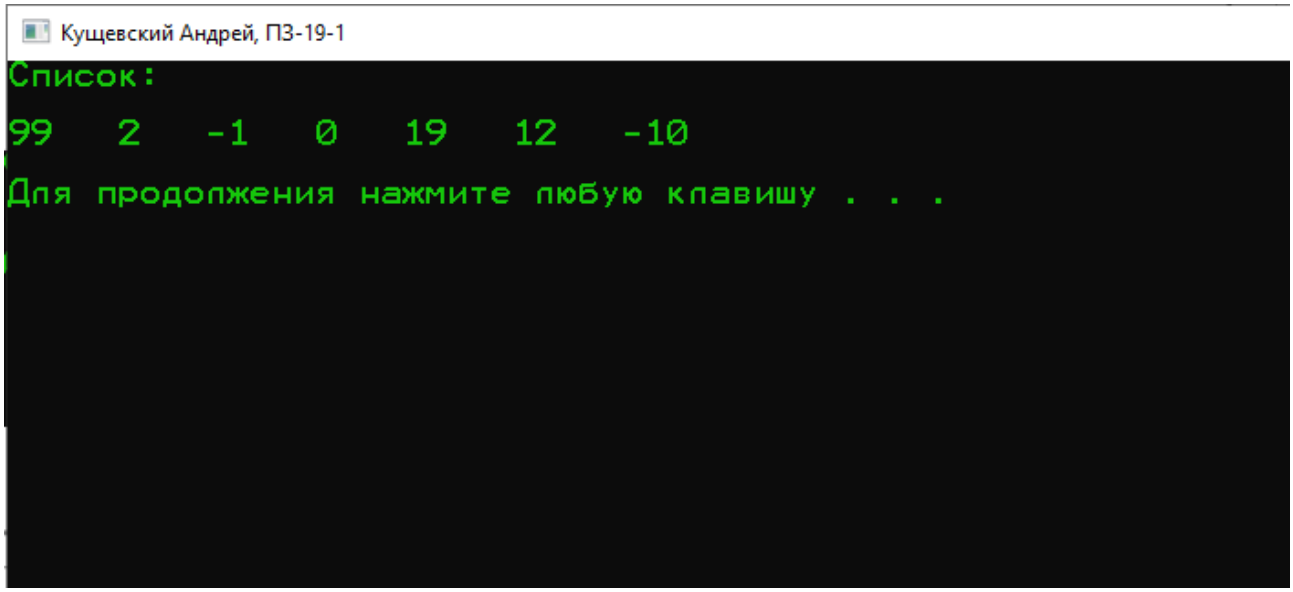
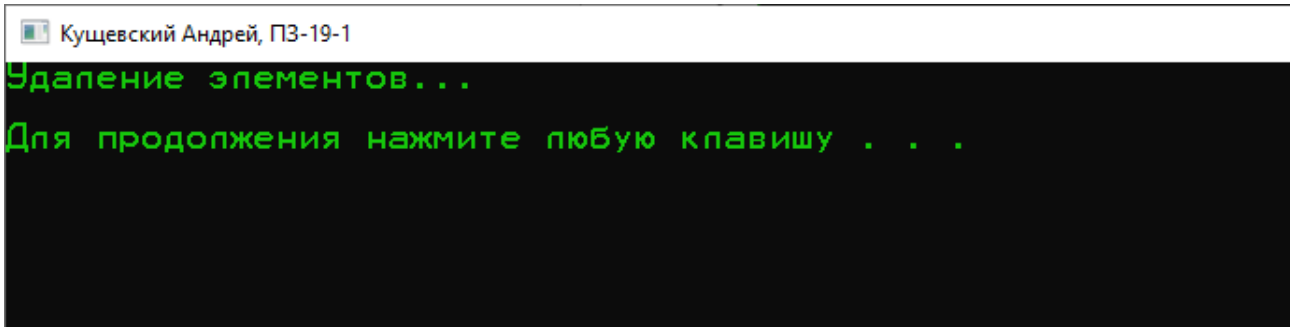


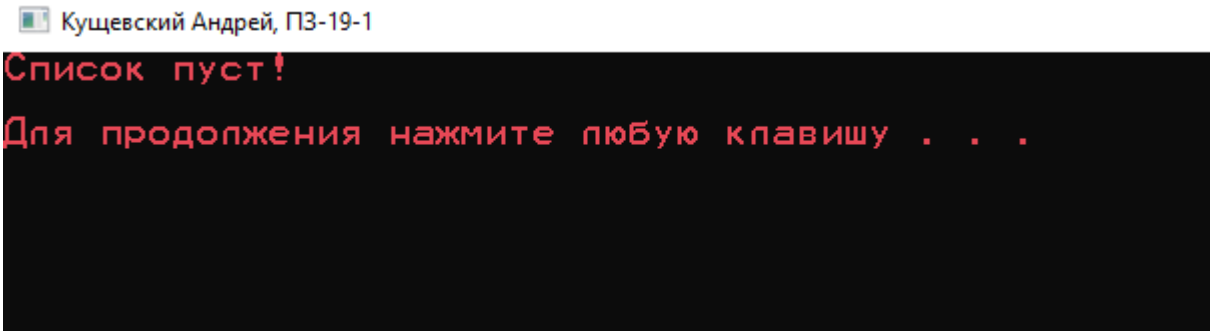
Рисунок 1.5 – Результат роботи функції calculateMinMax()





Рисунки 1.6 – 1.7 – Результат роботи методу removeMin()





Рисунки 1.8 – 1.9 – Результат роботи методу removeList()

Висновок: отримав практичні навички в формуванні та обробці динамічної структури даних – однозв’зяний список.

ЛАБОРАТОРНА РОБОТА № 3-4

Розробка основних функцій обробки черги та стеку. Складання та налагодження програм обробки черги та стеку.

Мета: отримання практичних навиків в формуванні та обробці «черги» та «стеку» на основі однозв’язного списку.

Хід роботи

2.1 Загальна постановка задачі

1 Провести аналіз поставленої задачі.

Загальна постановка завдання:

Скласти та налагодити програму обробки «черги» та «стеку» на основі однозв’язного списку за алгоритмом згідно Вашого варіанту. Програма повинна задовольняти наступним вимогам:

організувати користувацьке меню, яке повинно містити наступні пункти:

1. Формування черги.
2. Обробка всієї черги. (Результатом роботи цього пункту повинно бути виведення результату обробки черги згідно Вашого варіанту).
3. Обробка всього стеку. (Результатом роботи цього пункту повинно бути виведення результату обробки стеку згідно Вашого варіанту).

– забезпечити коректне введення користувачем вхідних даних;

– при обробці черги та стеку враховувати, що шукані елементи можуть бути відсутні. В цьому випадку вивести користувачеві відповідне повідомлення;

– введення та виведення вхідних та вихідних даних повинно містити необхідні для користувача повідомлення.

2 Розробити та налагодити програму рішення задачі.

3 Оформити звіт з лабораторної роботи.

2.2 Постановка задачі за варіантом

9. Знайти суму елементів черги з діапазону [x;y] (x та y вводяться користувачем). Ті елементи черги, які не належать діапазону [x;y], помістити в стек. Підрахувати кількість додатних елементів в стеку.

2.3 Розробка головної функції

Лістинг 2.1 – Текст програми

```
#include <iostream>
#include <Windows.h>
#include <conio.h>
#include <vector>
#include <string>
#define UP 72
#define DOWN 80
#define ENTER 13

using namespace std;

HANDLE hStdOut = GetStdHandle(STD_OUTPUT_HANDLE);

void gotoxy(short x, short y) {

  SetConsoleCursorPosition(hStdOut, { x, y });

}

void hideCursor(bool show) {

  CONSOLE_CURSOR_INFO info;
  info.bVisible = show;
  info.dwSize = 20;
  SetConsoleCursorInfo(hStdOut, &info);

}
```

```
    void clearCinBuff() {

      cin.sync();
      cin.clear();
      cin.ignore(cin.rdbuf()->in_avail());

    }

    //реализация стека
    struct Stack {

      //узел стека
      struct Node{

         float data;
         Node* next;//указатель на следующий узел стека

         Node(float value) : data(value) {}
         Node(float value, Node* node) : data(value), next(node)
{}
      };

      Node* _top;//верхний узел стека(корневой)
      int _size; //размер стека

      Stack() : _top(0) {}

      //метод для добавления элемента в конец стека
      void push(float data) {

         //если верхний элемент стека не nullptr(присутсвуют
какие-либо значения)
         if (_top != nullptr) {
```

```
            Node* temp = new Node(data, _top);//создается новый
узел, указатель в новом узле будет указывать тот узел, который
перед ним
            _top = temp;


        }else{

            _top = new Node(data);//новый узел становится
верхушкой стека

        }

        ++_size;

    }

    //метод для удаления последнего элемента в стеке(top)
    void pop() {

        //если указатель на верхний элемент == nullptr, то
прерывание работы метода
        if (_top == nullptr) {
            return;
        }
        else {

            Node* temp = _top;
            _top = _top->next;//элемент, который стоял после
удаляемого становится корневым элементом
            delete temp;
            --_size;

        }

    }
```

```
    //метод для вывода последнего(корневого) элемента на экран
    float& top() {

        return _top->data;

    }

    //метод для проверки пуст ли стек
    bool is_empty() {

        if(_size == 0) {
            return true;
        }
        else {
            return false;
        }

    }

    //метод для возврата количества элементов в стеке
    int& size() {
        return _size;
    }

};

//реализация очереди
struct Queue {

    //узел очереди
    struct Node {

        float data;
        Node* next;
```

```
        Node(float value, Node* node) : data(value), next(node)
{}

    };

    Node* head;
    Node* tail;
    int _size;

    //метод для возврата количества элементов в очереди
    int& size() {

        return _size;

    }

    //метод проверки на пустоту очереди
    bool is_empty() {

        if (_size == 0) {
            return true;
        }
        else {
            return false;
        }

    }

    //метод добавления элемента в очередь
    void push(float value) {

        Node* node = new Node(value, nullptr);//передаем в
конструктор значение для поля дата, и указатель nullptr т.к
элемент будет располагаться в конце очереди
```

```
        //если очередь пуста, то элемент становиться и хвостом и
головой очереди
        if (_size == 0) {

            head = node;
            tail = node;

        }
        else {

            tail->next = node;
            tail = node;

        }

        ++_size;

    }

    //метод для извлечения элемента из очереди
    float pop() {

        float value = head->data;

        if (head == tail) {

            head = nullptr;
            tail = nullptr;

        }
        else {

            head = head->next;

        }
```

```
            --_size;

            return value;

        }

        //метод для получения значения первого элемента в
очереди(head)
        float front() {

            return head->data;

        }

        //метод для получения значения последнего элемента в
очереди(tail)
        float back() {

            return tail->data;

        }

    };

    Stack stack;
    Queue queue;

    void showQueue() {

      system("cls");
      SetConsoleTextAttribute(hStdOut, FOREGROUND_GREEN |
FOREGROUND_INTENSITY);

      if (queue.is_empty()) {
         SetConsoleTextAttribute(hStdOut, FOREGROUND_RED |
FOREGROUND_INTENSITY);
```

```
            cout << "Очередь пуста!\n\n";
            system("pause");
            system("cls");
            SetConsoleTextAttribute(hStdOut, FOREGROUND_GREEN |
FOREGROUND_INTENSITY);
            return;
        }

        cout << "Очередь:" << endl << endl;
        while (!queue.is_empty()) {

            cout << queue.pop() << "     ";

        }

        cout << endl << endl;
        hideCursor(false);
        system("pause");
        system("cls");
        return;

    }

    void showStack() {

        system("cls");
        SetConsoleTextAttribute(hStdOut, FOREGROUND_GREEN |
FOREGROUND_INTENSITY);

        if (stack.is_empty()) {
            SetConsoleTextAttribute(hStdOut, FOREGROUND_RED |
FOREGROUND_INTENSITY);
            cout << "Стек пуст!\n\n";
            system("pause");
            system("cls");
```

```
            SetConsoleTextAttribute(hStdOut, FOREGROUND_GREEN |
FOREGROUND_INTENSITY);
            return;
        }

        cout << "Стек:" << endl << endl;
        while (!stack.is_empty()) {

            cout << stack.top() << "     ";
            stack.pop();

        }

        cout << endl << endl;
        hideCursor(false);
        system("pause");
        system("cls");
        return;

    }

    void addToQueue() {

        system("cls");
        SetConsoleTextAttribute(hStdOut, FOREGROUND_GREEN |
FOREGROUND_INTENSITY);

        hideCursor(true);

        if (!(queue.is_empty())) {
            cout << "В данный момент в очереди присутствуют
значения!" << endl << endl;
        }

        cout << "Введите количество элементов, которые желаете
добавить в очередь:";
```

```
        int size;
        while (!(cin >> size) || (size <= 0)) {
           clearCinBuff();
           SetConsoleTextAttribute(hStdOut, FOREGROUND_RED |
FOREGROUND_INTENSITY);
           cout << "Ошибка! Введите корректное значение:";
        }
        SetConsoleTextAttribute(hStdOut, FOREGROUND_GREEN |
FOREGROUND_INTENSITY);

        float data;

        cout << endl;

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

           cout << "[" << i + 1 << "]:";
           while (!(cin >> data)) {
                clearCinBuff();
                SetConsoleTextAttribute(hStdOut, FOREGROUND_RED |
FOREGROUND_INTENSITY);
                cout << "Ошибка! Введите корректное значение для ["
<< i + 1 << "]:";
           }

           SetConsoleTextAttribute(hStdOut, FOREGROUND_GREEN |
FOREGROUND_INTENSITY);

           queue.push(data);

        }

        cout << endl;
        hideCursor(false);
        system("pause");
        system("cls");
```

```
        return;

    }

    void findSumQueue() {

        system("cls");
        SetConsoleTextAttribute(hStdOut, FOREGROUND_GREEN |
FOREGROUND_INTENSITY);

        if (queue.is_empty()) {
            SetConsoleTextAttribute(hStdOut, FOREGROUND_RED |
FOREGROUND_INTENSITY);
            cout << "Очередь пуста!\n\n";
            system("pause");
            system("cls");
            SetConsoleTextAttribute(hStdOut, FOREGROUND_RED |
FOREGROUND_INTENSITY);
            return;
        }

        hideCursor(true);

        cout << "Введите x:";
        float x;
        while (!(cin >> x)) {

            clearCinBuff();
            SetConsoleTextAttribute(hStdOut, FOREGROUND_RED |
FOREGROUND_INTENSITY);
            cout << "Ошибка! Введите корректное значение:";

        }
        SetConsoleTextAttribute(hStdOut, FOREGROUND_GREEN |
FOREGROUND_INTENSITY);
```

```
        cout << endl << "Введите y:";
        float y;
        while (!(cin >> y)) {

            clearCinBuff();
            SetConsoleTextAttribute(hStdOut, FOREGROUND_RED |
FOREGROUND_INTENSITY);
            cout << "Ошибка! Введите корректное значение:";

        }
        SetConsoleTextAttribute(hStdOut, FOREGROUND_GREEN |
FOREGROUND_INTENSITY);

        float sum = 0;
        float temp = 0;

        vector<float> toStack;
        vector<float> queueValues;

        while (!queue.is_empty()) {

            temp = queue.pop();
            queueValues.push_back(temp);

        }

        for (int i = 0; i < queueValues.size(); ++i) {

            if ((queueValues[i] >= x && queueValues[i] <= y) ||
(queueValues[i] <= x && queueValues[i] >= y)) {

                sum += queueValues[i];

            }
            else {
                toStack.push_back(queueValues[i]);
```

```
            }

        }

        cout << endl << endl << "Сумма элементов очереди из
диапазона[" << x << "," << y << "]:" << sum;

        cout << endl << endl << endl << "Следующие элементы будут
помещены в стек:" << endl << endl;
        for (int i = 0; i < toStack.size(); ++i) {

            cout << toStack[i] << "    ";
            stack.push(toStack[i]);

        }

        cout << endl << endl << endl;
        hideCursor(false);
        system("pause");
        system("cls");
        return;

    }

    void amountPositiveInStack() {

        system("cls");
        SetConsoleTextAttribute(hStdOut, FOREGROUND_GREEN |
FOREGROUND_INTENSITY);

        if (stack.is_empty()) {
            SetConsoleTextAttribute(hStdOut, FOREGROUND_RED |
FOREGROUND_INTENSITY);
            cout << "Стек пуст!\n\n";
            system("pause");
            system("cls");
```

```
            SetConsoleTextAttribute(hStdOut, FOREGROUND_GREEN |
FOREGROUND_INTENSITY);
            return;
        }

        int amountPositive = 0;
        float temp = 0;

        vector<float> stackValues;

        do {

            temp = stack.top();
            stackValues.push_back(temp);

            if (temp > 0) {
                 ++amountPositive;
            }
            stack.pop();

        } while (!stack.is_empty());

        cout << "Стек:" << endl << endl;
        for (int i = 0; i < stackValues.size(); ++i) {

            cout << stackValues[i] <<  "    ";

        }

        cout << endl << endl << endl << "Количество положительных
элементов стека:" << amountPositive;

        cout << endl << endl << endl;
        hideCursor(false);
        system("pause");
        system("cls");
```

```
        return;

    }

    void mainMenu() {

        SetConsoleTextAttribute(hStdOut, FOREGROUND_GREEN |
FOREGROUND_INTENSITY);
        hideCursor(false);

        string mainMenu[] =
        {
           "Просмотр состояния очереди",
           "Просмотр состояния стека",
           "Добавление элементов в очередь",
           "Расчет суммы элементов очереди из диапазона [x,y]",
           "Расчет количества положительных элементов стека",
           "Выход из программы"
        };

        short x = 0;
        short y = 0;

        int activeMenu = 0;
        int key = 0;

        while (true) {
           hideCursor(false);
           x = 10, y = 3;

           for (int i = 0; i < size(mainMenu); ++i) {

                if (activeMenu == i) {
                     SetConsoleTextAttribute(hStdOut,
FOREGROUND_GREEN | FOREGROUND_INTENSITY);
                }
```

```
                else {
                    SetConsoleTextAttribute(hStdOut,
FOREGROUND_GREEN);
                }

                gotoxy(x, ++y);
                cout << mainMenu[i];

            }

            key = _getch();

            switch (key) {

            case UP: {
                if (activeMenu > 0) {
                    --activeMenu;
                }
                break;
            }

            case DOWN: {
                if (activeMenu < size(mainMenu) - 1) {
                    ++activeMenu;
                }
                break;
            }

            case ENTER: {
                switch (activeMenu) {

                case 0: {
                    showQueue();
                    break;
                }
```

```
                case 1: {
                     showStack();
                     break;
                }

                case 2: {
                     addToQueue();
                     break;
                }

                case 3: {
                     findSumQueue();
                     break;
                }

                case 4: {
                     amountPositiveInStack();
                     break;
                }

                case 5: {
                     system("cls");
                     SetConsoleTextAttribute(hStdOut, FOREGROUND_RED
| FOREGROUND_INTENSITY);
                     cout << "Завершение работы программы...\n\n";
                     system("pause");
                     exit(0);
                     break;
                }

                }
                break;
           }
           }

        }
```

```
}

int main() {

  setlocale(0, "rus");
  SetConsoleTitle(L"Кущевский Андрей, ПЗ-19-1");

  mainMenu();

  _getch();
  return 0;

}
```

Вікна виконання програми дивитись на рисунках 3.1 – 3.7:

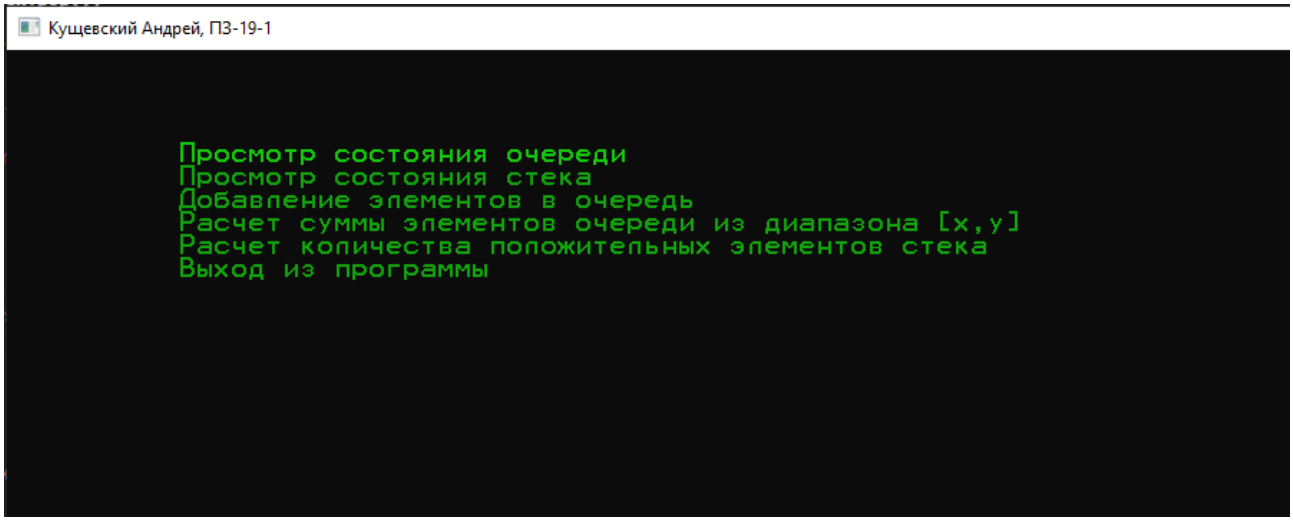


Рисунок 3.1 – Головне меню програми

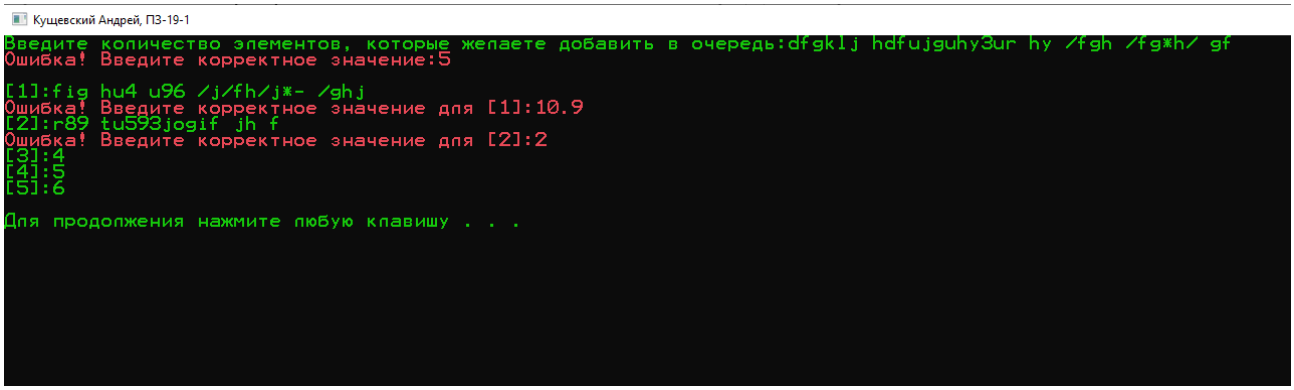


Рисунок 3.2 – Перевірка користувацького друку

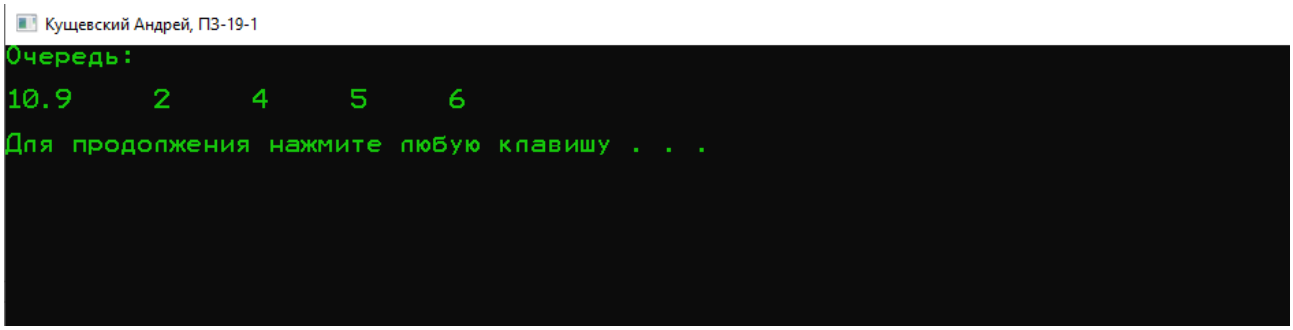


Рисунок 3.3 – Результат роботи функції showQueue()

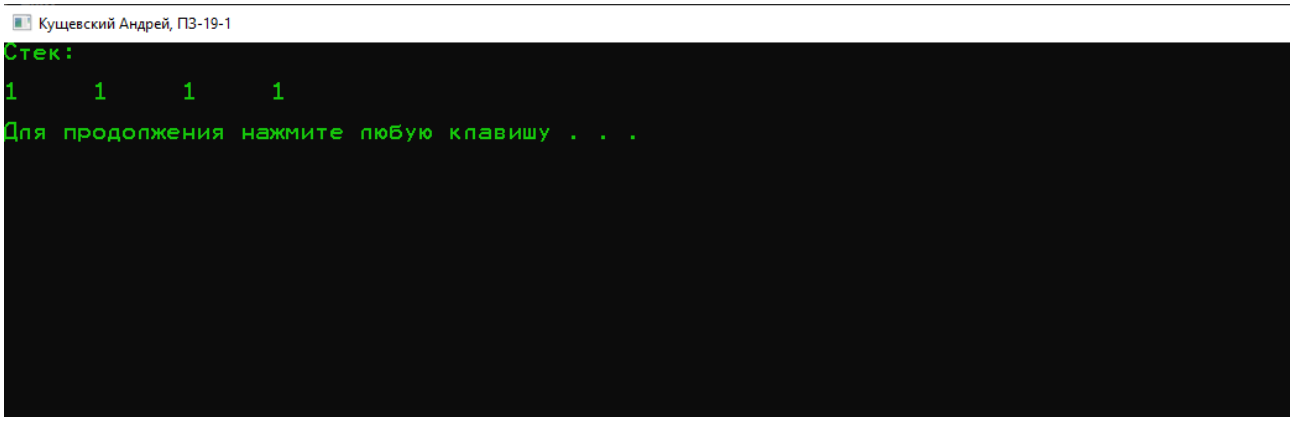


Рисунок 3.4 – Результат роботи функції showStack()

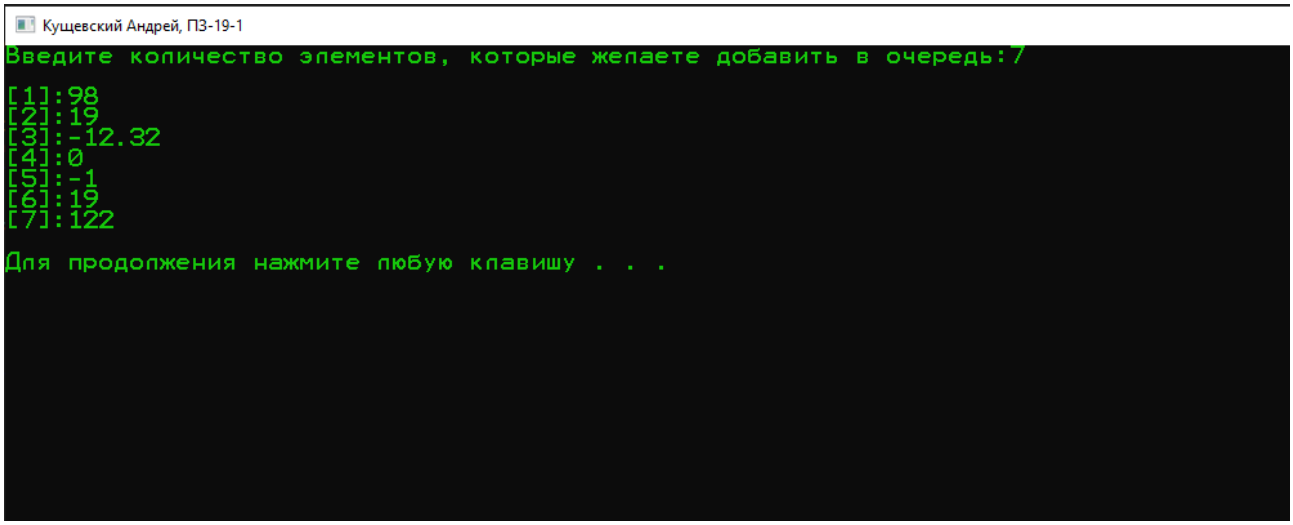


Рисунок 3.5 – Результат роботи функції addToQueue()

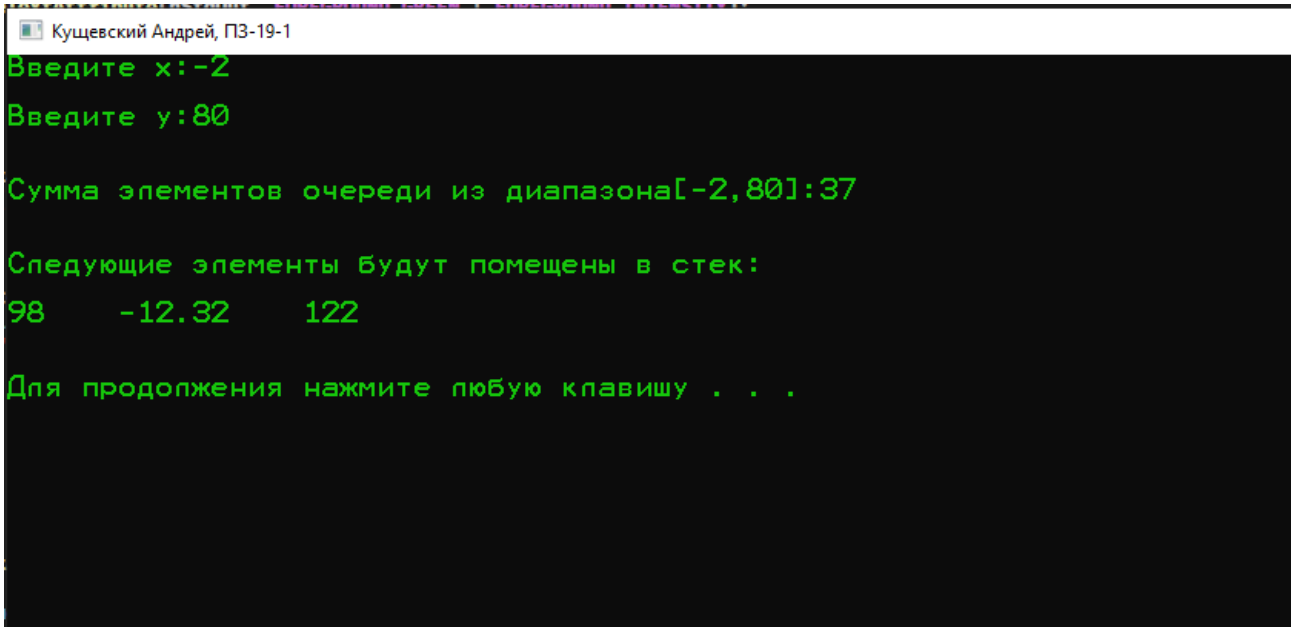


Рисунок 3.6 – Результат роботи функції findSumQueue()

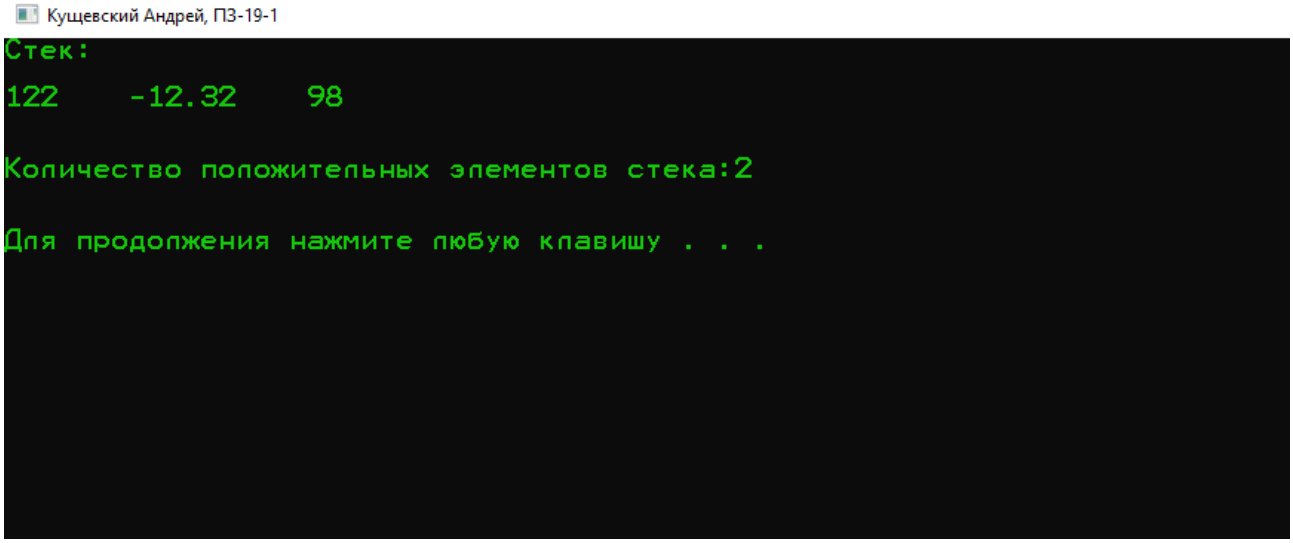


Рисунок 3.7 – Результат роботи методу amountPosistiveInStack()

Висновок: отримав практичних навичків в формуванні та обробці «черги» та «стеку» на основі однозв’язного списку.

ЛАБОРАТОРНА РОБОТА № 5-6

Складання та налагодження програми рішення задачі створення та обходу бінарних дерев.

Мета: отримання навиків в організації динамічної структури даних – бінарного дерева.

Хід роботи

5.1 Загальна постановка задачі

1 Провести аналіз поставленої задачі.

Загальна постановка завдання:

Скласти та налагодити програму створення та обробки бінарного дерева на основі нелінійного списку за алгоритмом згідно Вашого варіанту. Програма повинна задовольняти наступним вимогам:

– організувати користувацьке меню, яке повинно містити наступні пункти:

1. Створення бінарного дерева.
2. Перегляд вмісту бінарного дерева.
3. Обробка дерева згідно Вашого варіанту.

– забезпечити коректне введення користувачем вхідних даних;

– при обробці черги та стеку враховувати, що шукані елементи можуть бути відсутні. В цьому випадку вивести користувачеві відповідне повідомлення;

– введення та виведення вхідних та вихідних даних повинно містити необхідні для користувача повідомлення.

2 Розробити та налагодити програму рішення задачі.

3 Оформити звіт з лабораторної роботи.

2.2 Постановка задачі за варіантом

22. Створити бінарне дерево дійсних чисел введенням даних з клавіатури. Використовуючи прямий обхід дерева, знайти мінімальний елемент дерева та вивести його на екран. Всі від’ємні числа дерева заміни мінімальним елементом. Вивести оновлене дерево на екран.

2.3 Розробка головної функції

Лістинг 5.1 – Текст програми

```
/*
  Вариант 22.
  Створити бінарне дерево дійсних чисел введенням даних з клавіатури.
  Використовуючи прямий обхід дерева, знайти мінімальний елемент дерева та вивести його на екран.
  Всі від’ємні числа дерева заміни мінімальним елементом. Вивести оновлене дерево на екран.
*/

#include <iostream>
#include <conio.h>
#include <windows.h>
#include <vector>
#include <string>
#include <iomanip>

#define UP 72
#define DOWN 80
#define ENTER 13

using namespace std;



int tabs = 0;
vector<float> vec;
```

```
    void hideCursor(bool showStatus)
    {

      _CONSOLE_CURSOR_INFO info;
      info.bVisible = showStatus;
      info.dwSize = 20;
      SetConsoleCursorInfo(GetStdHandle(STD_OUTPUT_HANDLE),
&info);

    }

    void gotoxy(int x, int y)
    {

      COORD coord;

      coord.X = x;
      coord.Y = y;
      SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE),
coord);

    }

    void setColor(int num)
    {

      SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE),
num);

    }

    void clearStreamInput()
    {

      cin.clear();
      cin.sync();
```

```
  cin.ignore(cin.rdbuf()->in_avail());

}

void endProcedure()
{

  clearStreamInput();
  cout << endl << endl;
  system("pause");
  system("cls");

}

float correctWrite()
{

  float value;
  setColor(14);

  while (!(cin >> value)) {
     setColor(12);
     clearStreamInput();
     cout << "Ошибка! Введите корректное значение:";

  }

  setColor(10);
  std::cout << std::endl << "Значение корректно!\n\n";
  setColor(14);

  return value;
}

struct Node {
```

```
        float data;
        Node *left;
        Node *right;

        Node(float num) : data(num), left(nullptr), right(nullptr)
{}
        Node() : data(), left(nullptr), right(nullptr) {}

    };

    struct BinaryTree {

        Node *head;

        void input() {

            char ans;

            std::cout << "Инициализация начального узла дерева:";
            head = new Node(correctWrite());

            std::cout << "Двигаемся влево? (Y - Да, другая клавиша -
отмена):";
            std::cin >> ans;
            head->left = Add(ans);

            std::cout << "\nДвигаемся вправо? (Y - Да, другая клавиша
- отмена):";
            std::cin >> ans;
            head->right = Add(ans);

        }

        bool isEmpty() {
            return head == nullptr;
        }
```

```
        Node *Add(char ans) {

            Node *work;

            if (ans == 'Y' || ans == 'y') {

                std::cout << std::endl << "Введите новый элемент: ";
                work = new Node(correctWrite());

                std::cout << "Добавить элемент влево?  (Y - Да,
другая клавиша - отмена):";
                std::cin >> ans;
                work->left = Add(ans);

                std::cout << "Добавить элемент вправо? (Y - Да,
другая клавиша - отмена):";
                std::cin >> ans;
                work->right = Add(ans);

                return work;

            }
            else {
                return nullptr;
            }

        }

        void replaceNegativeOnMin(Node *(&node), float min) {

            if (!node) {

                return;

            }
```

```
        if (node->data < 0) node->data = min;
        replaceNegativeOnMin(node->left, min);
        replaceNegativeOnMin(node->right, min);


    }

    void findMin(Node *p) {

        Node *work;
        work = p;

        if (!p) {

            return;

        }

        vec.push_back(work->data);
        if (work->left != nullptr) findMin(work->left);
        if (work->right != nullptr) findMin(work->right);

    }

};

BinaryTree tree;

//реализация рекурсивного вывода дерева
void outputTree(Node *(&Node))
{

    //если в указатель nullptr, прекращение работы функции
    if (tree.isEmpty()) {
```

```
            return;

        }

        //если ветка существует то увеличение счетчика рекурсивно
вызванных процедур, который
        //будет считать отступы для красивого вывода дерева
        tabs += 5;

        //вывод ветки и её подветок справа
        if (Node->right != nullptr) outputTree(Node->right);

        for (int i = 0; i < tabs; ++i) cout << " ";//отступы
        cout << Node->data << endl;//данный этой ветки

        //вывод ветки и её подветок слева
        if (Node->left != nullptr) outputTree(Node->left);

        tabs -= 5;//уменьшение количества отступов

        return;

    }

    void clearTree(Node *(&p))
    {

        //если в дереве отсутствуют ноды, прекращение работы функции
        if (tree.isEmpty()) {
            return;
        }

        Node *work;
        work = p;

        if (work->left != nullptr)  clearTree(work->left);
```

```
        if (work->right != nullptr) clearTree(work->right);
        delete work;

        p = nullptr;

        return;

    }

    /*************************************************************
*************************************************************/
    //start func menu

    //1 pos menu
    void showTreeMenu()
    {

        system("cls");
        hideCursor(false);

        if (!tree.head) {

            setColor(12);
            cout << "В дереве отсутствуют элементы, воспользуйтесь
соответствующим пунктом меню для инициализации дерева!";
            endProcedure();
            return;

        }

        setColor(14);
        outputTree(tree.head);

        endProcedure();
        return;
```

```
    }

    //2 pos menu
    void createTreeMenu()
    {

      system("cls");

      if (tree.isEmpty()) {
         setColor(12);
         cout << "В данный момент в дереве отсутствуют элементы!"
<< endl << endl;
         setColor(14);
      }
      else {
         setColor(12);
         cout << "Дерево ранее уже было инициализировано(данные
будут удалены)!" << endl << endl;
         setColor(14);
      }

      hideCursor(true);

      tree.input();

      hideCursor(false);
      cout << endl;
      system("pause");
      system("cls");
      return;

    }

    //3 pos menu
    void findMinMenu()
    {
```

```
        system("cls");
        setColor(14);

        if (!tree.head) {

           setColor(12);
           cout << "В дереве отсутствуют элементы, воспользуйтесь
соответствующим пунктом меню для инициализации дерева!";
           endProcedure();
           return;

        }

        tree.findMin(tree.head);

        cout << "Элементы:" << endl << endl;
        for (int i = 0; i < vec.size(); ++i) {

           cout << setw(4) << vec[i];

        }

        float min = vec[0];

        for (int i = 0; i < vec.size(); ++i) {

           if (vec[i] < min) {

                min = vec[i];

           }

        }

        cout << endl << endl << "Минимальный элемент равен:" << min;
```

```
        vec.clear();

        endProcedure();
        return;

    }

    //4 pos menu
    void negativeReplaceMenu()
    {

        system("cls");
        hideCursor(false);

        if (!tree.head) {

            setColor(12);
            cout << "В дереве отсутствуют элементы, воспользуйтесь
соответствующим пунктом меню для инициализации дерева!";
            endProcedure();
            return;

        }

        cout << "Видоизменённое дерево после замены отрицательных
элементов на минимальное:" << endl << endl;

        tree.findMin(tree.head);
        float min = vec[0];
        unsigned int indexMin = 0;

        for (int i = 0; i < vec.size(); ++i) {
            if (vec[i] < min) {

                min = vec[i];
```

```
            }
        }

        tree.replaceNegativeOnMin(tree.head, min);

        outputTree(tree.head);

        vec.clear();
        hideCursor(false);
        endProcedure();
        return;

    }

    //5 pos menu
    void clearTreeMenu()
    {

        system("cls");
        setColor(14);

        if (tree.isEmpty()) {
            setColor(12);
            cout << "В дереве отсутствуют элементы, воспользуйтесь
соответствующим пунктом меню для инициализации дерева!";
            endProcedure();
            return;
        }

        clearTree(tree.head);
        cout << "Очистка дерева...";

        endProcedure();
        return;
```

```
    }

    //end func menu
    /*****************************************************************
*****************************************************************/

    void mainMenu()
    {

      std::string mainMenu[] =
      {
         "Просмотр дерева",
         "Инициализация дерева значениями",
         "Поиск минимального значения",
         "Замена отрицательных элементов минимальным",
         "Очистка дерева",
         "Выход из программы"
      };

      short x = 0;
      short y = 0;
      int key = 0;
      int activeMenu = 0;

      while (true) {
         hideCursor(false);

         x = 10;
         y = 3;

         for (int i = 0; i < size(mainMenu); ++i) {

              if (activeMenu == i) {
                   setColor(13);
              }
              else {
```

```
                setColor(14);
            }

            gotoxy(x, ++y);
            std::cout << mainMenu[i];

        }

        key = _getch();

        switch (key) {

        case UP: {
            if (activeMenu > 0) {
                --activeMenu;
            }
            else {
                activeMenu = size(mainMenu) - 1;
            }

            break;
        }
        case DOWN: {

            if (activeMenu < size(mainMenu) - 1) {
                ++activeMenu;
            }
            else {
                activeMenu = 0;
            }

            break;
        }

        case ENTER: {
            switch (activeMenu) {
```

```
                case 0: {
                     showTreeMenu();
                     break;
                }

                case 1: {
                     createTreeMenu();
                     break;
                }

                case 2: {
                     findMinMenu();
                     break;
                }

                case 3: {
                     negativeReplaceMenu();
                     break;
                }

                case 4: {
                     clearTreeMenu();
                     break;
                }

                case 5: {
                     system("cls");
                     setColor(14);
                     std::cout << "Завершение работы программы..."
<< std::endl << std::endl;
                     system("pause");
                     return;
                }

                }
                break;
```

```
        }

        }

    }

}

int main() {

    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);
    SetConsoleTitle(L"Кущевский Андрей, ПЗ-19-1");

    mainMenu();
    clearTree(tree.head);

    return 0;

}
```

Вікна виконання програми дивитись на рисунках 5.1 – 5.7:

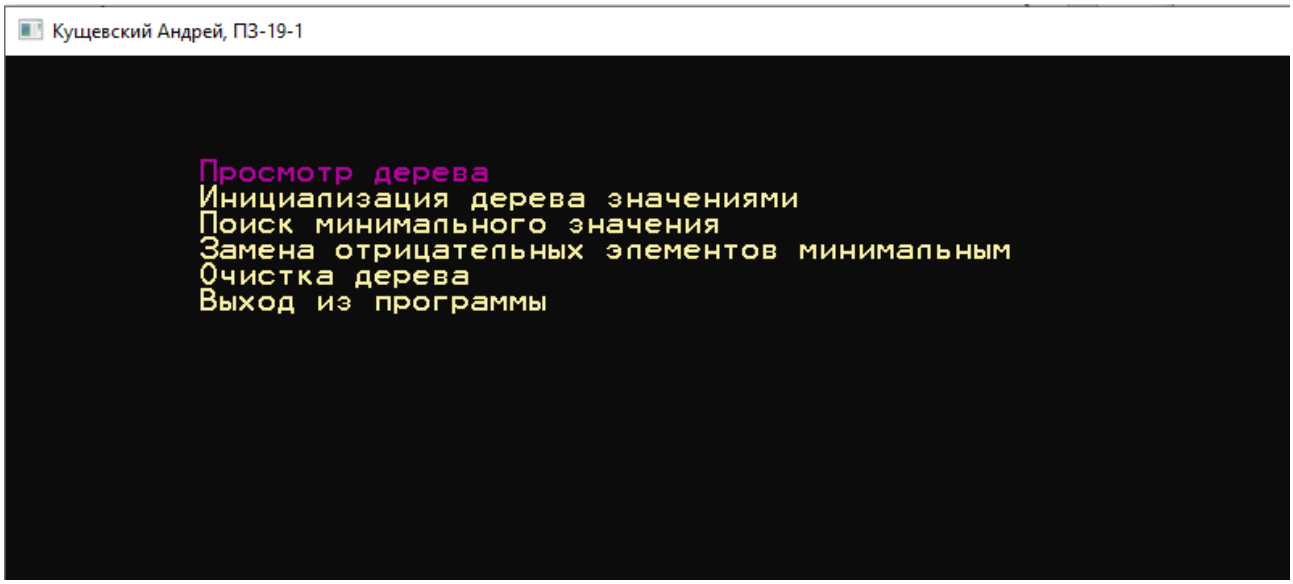


Рисунок 5.1 – Головне меню програми

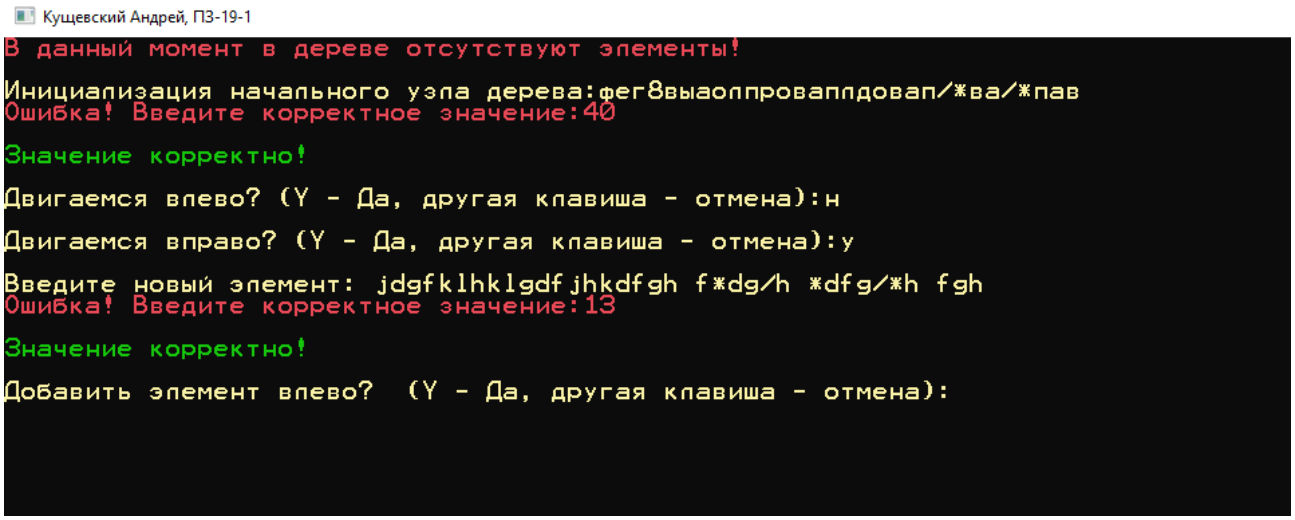


Рисунок 5.2 – Перевірка користувацького друку

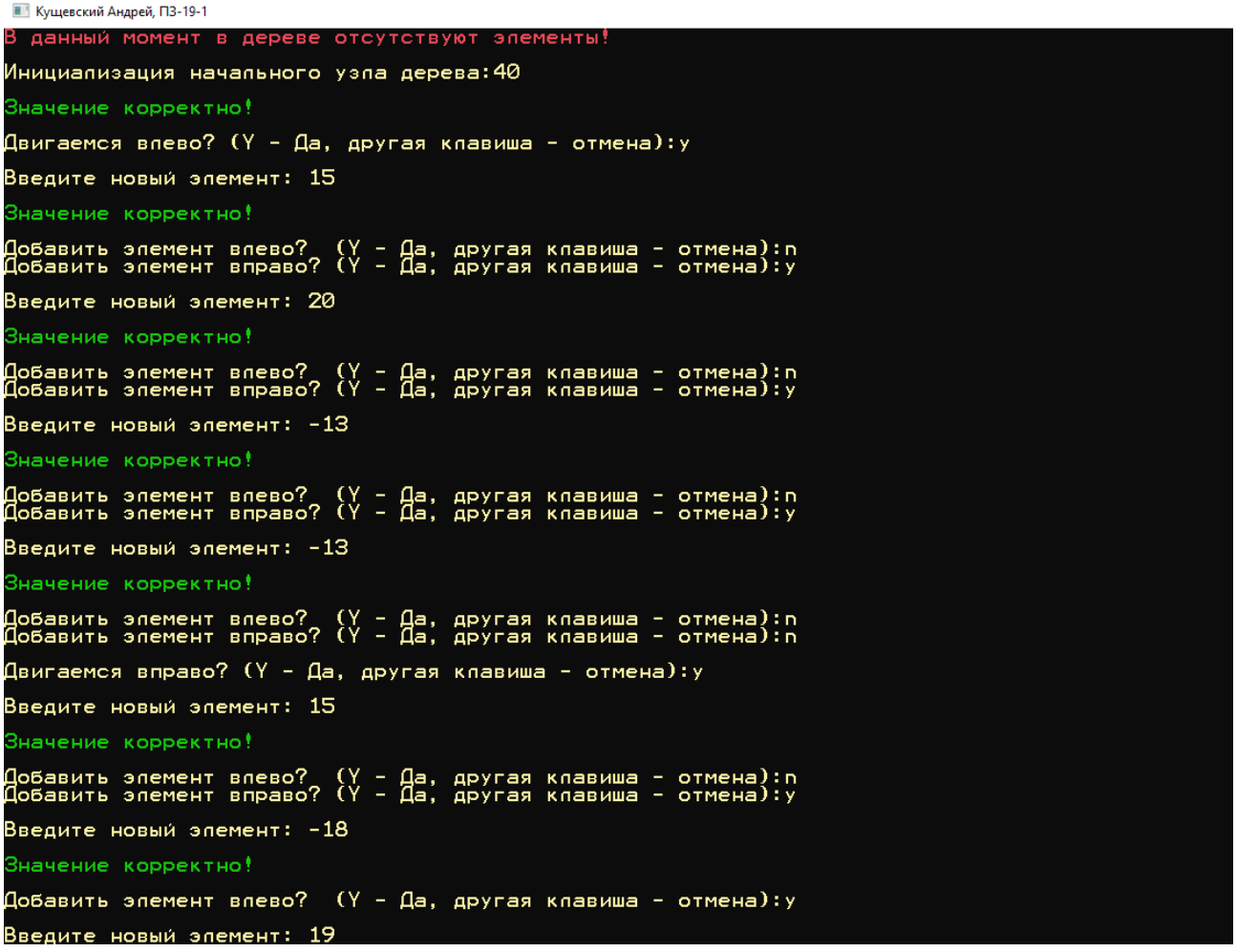


Рисунок 5.3 – Результат роботи функції createTreeMenu()

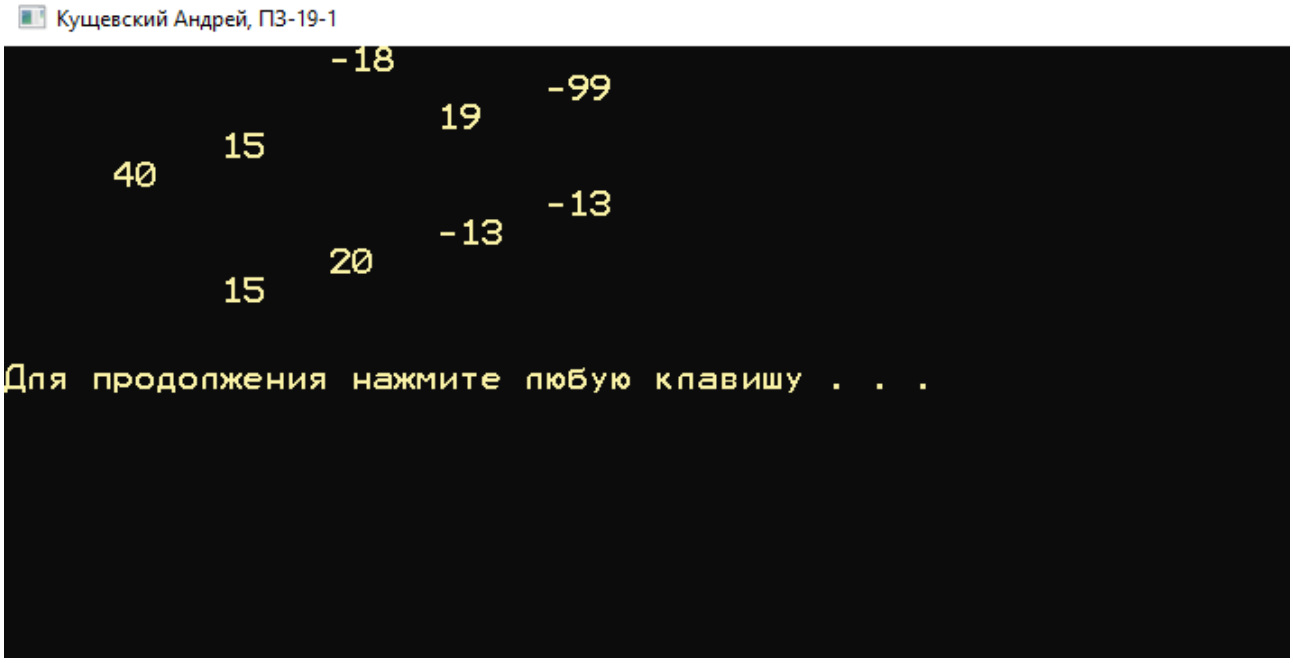


Рисунок 5.4 – Результат роботи функції outputTreeMenu()

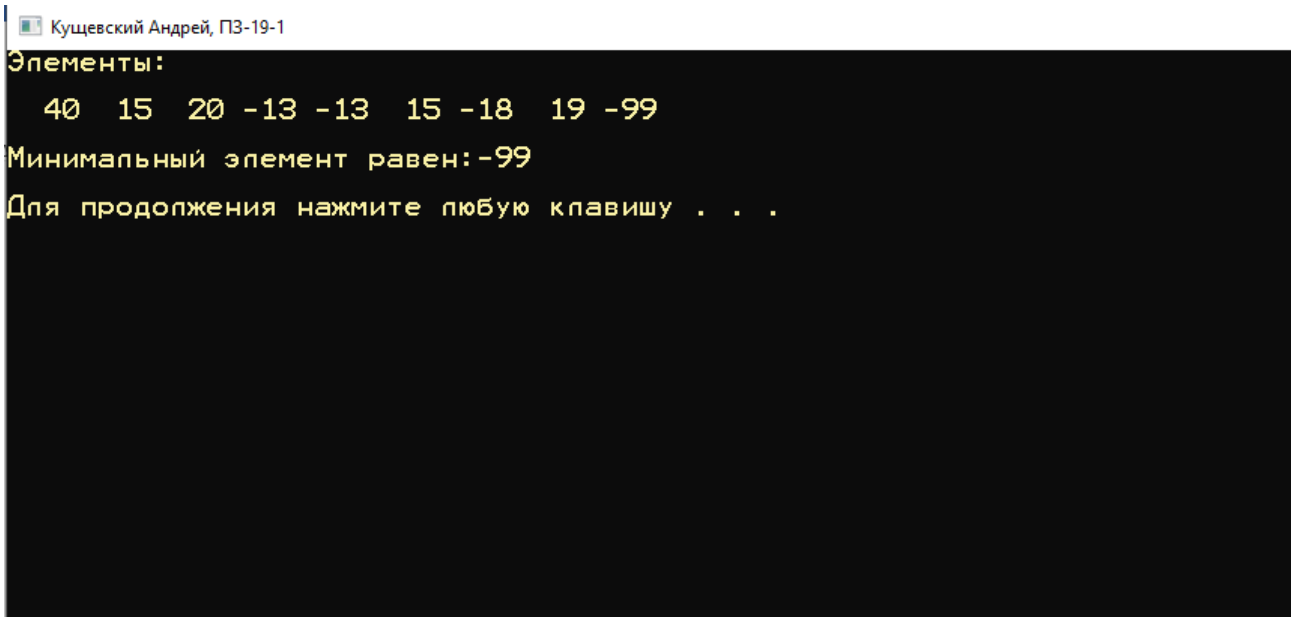


Рисунок 5.5 – Результат роботи функції findMinMenu()

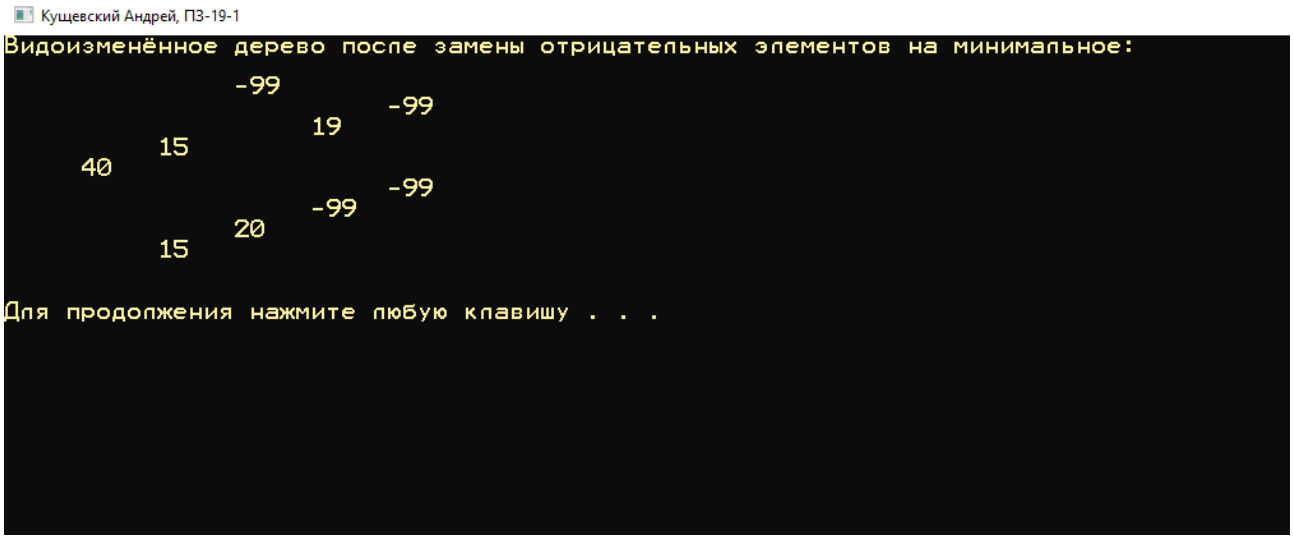
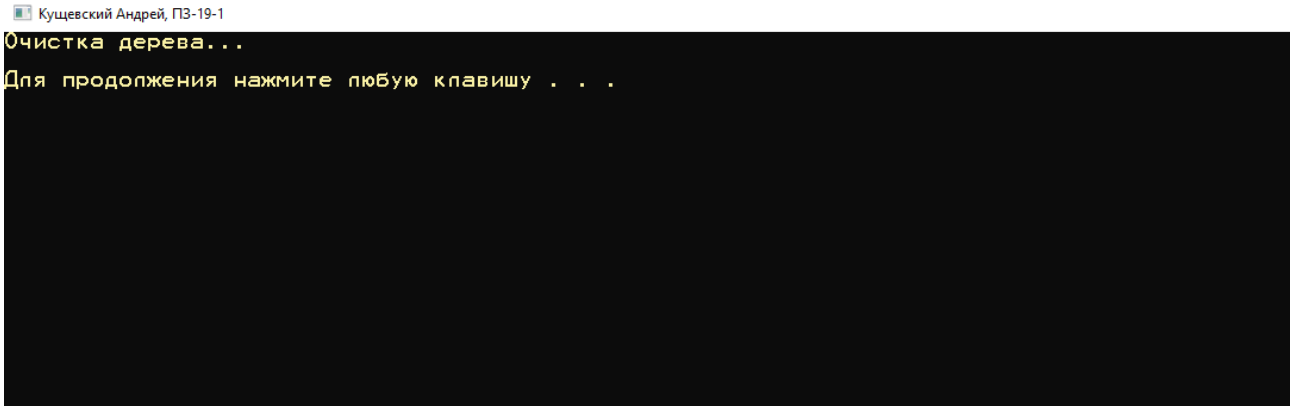


Рисунок 5.6 – Результат роботи функції replaceNegativeOnMin()



Рисунки 5.7 – Результат роботи функції clearTree()

Висновок: отримав навичків в організації динамічної структури даних – бінарного дерева.

ЛАБОРАТОРНА РОБОТА № 7-8

Складання та налагодження програм реалізації алгоритмів сортування елементів масиву даних.

Мета: здобуття навичок роботи зі структурованим типом даних масив та реалізації алгоритмів внутрішнього сортування.

Хід роботи

7.1 Загальна постановка задачі

Скласти та налагодити програму створення та сортування матриці за алгоритмом згідно Вашого варіанту. Програма повинна задовольняти наступним вимогам:

– організувати користувацьке меню, яке повинно містити наступні пункти:

1. Створення матриці.
2. Виведення матриці.
3. Обробка матриці згідно Вашого варіанту.

– забезпечити коректне введення користувачем вхідних даних. Тип матриці – динамічна або псевдодинамічна – обрати самостійно;

– введення та виведення вхідних та вихідних даних повинно містити необхідні для користувача повідомлення.

1 Розробити та налагодити програму рішення задачі.

2 Оформити звіт з лабораторної роботи.

3 Розробити та налагодити програму рішення задачі.

4 Оформити звіт з лабораторної роботи.

7.2 Постановка задачі за варіантом

9 Дано двовимірний масив дійсних чисел a[1..n,1..m] (кількість рядків n та кількість стовпців m вводиться користувачем). Впорядкувати за

спаданням k-й та p-й рядки масиву використовуючи алгоритм шейкерного сортування (k та p вводяться користувачем).

7.3 Розробка головної функції

Лістинг 7.1 – Текст програми

```
//Кущевский Андрей, ПЗ-19-1

/*
  Дано двовимірний масив дійсних чисел  a[1..n,1..m] (кількість рядків n та кількість стовпців m вводиться користувачем).
  Впорядкувати за спаданням k-й та p-й рядки масиву використовуючи алгоритм шейкерного сортування (k та  p  вводяться користувачем).
*/

#include <iostream>
#include <windows.h>
#include <conio.h>
#include <string>
#include <iomanip>
#define UP 72
#define DOWN 80
#define ENTER 13

using namespace std;

HANDLE hStdOut = GetStdHandle(STD_OUTPUT_HANDLE);

void hideCursor(bool show) {

  CONSOLE_CURSOR_INFO info;
  info.bVisible = show;
  info.dwSize = 10;
```

```
    SetConsoleCursorInfo(hStdOut, &info);

  }

  void gotoxy(short x, short y) {

    SetConsoleCursorPosition(hStdOut, { x, y });

  }

  void normalText() {

    SetConsoleTextAttribute(hStdOut, FOREGROUND_GREEN |
FOREGROUND_INTENSITY);

  }

  void errorText() {

    SetConsoleTextAttribute(hStdOut, FOREGROUND_RED |
FOREGROUND_INTENSITY);

  }

  void clearCinBuff() {

    cin.clear();
    cin.sync();
    cin.ignore(cin.rdbuf()->in_avail());

  }

  void sizeOfMatrix(int(&rows), int(&cols)) {

    normalText();
    hideCursor(true);
```

```
    cout << "Введите количество строк:";
    while (!(cin >> rows) || (rows <= 0)) {

        clearCinBuff();
        errorText();
        cout << "Ошибка! Введите корректное значение:";

    }
    normalText();

    cout << endl << "Введите количество столбцов:";
    while (!(cin >> cols) || (cols <= 0)) {

        clearCinBuff();
        errorText();
        cout << "Ошибка! Введите корректное значение:";

    }
    normalText();

}

void outputMatrix(float **matrix, int rows, int cols) {

    system("cls");
    normalText();
    hideCursor(false);

    if (rows == 0) {
        cout << "В матрице отсутствуют значения!";
        cout << endl << endl;
        system("pause");
        system("cls");
        return;
    }
```

```
    cout << "Матрица:" << endl;

    for (int i = 0; i < rows; ++i) {
       for (int j = 0; j < cols; ++j) {

            cout << matrix[i][j] << "\t";

       }
       cout << endl;
    }

    cout << endl << endl;
    system("pause");
    system("cls");

}

void inputMatrix(float **(&matrix), int(&rows), int(&cols)) {

    system("cls");
    hideCursor(true);
    normalText();

    if (rows != 0) {

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

            delete matrix[i];

       }

       delete[] matrix;

    }
```

```
    sizeOfMatrix(rows, cols);

    matrix = new float *[rows];
    for (int i = 0; i < rows; ++i) {

       matrix[i] = new float[cols];

    }

    cout << endl << endl;

    for (int i = 0; i < rows; ++i) {
       for (int j = 0; j < cols; ++j) {

            cout << "[" << i + 1 << "][" << j + 1 << "]:";
            while (!(cin >> matrix[i][j])) {

                 clearCinBuff();
                 errorText();
                 cout << "Ошибка! Введите корректное значение:";

            }
            normalText();

       }
    }


    cout << endl << endl;
    hideCursor(false);
    system("pause");
    system("cls");

}
```

```
    void cocktailSortSelectRow(float **(&matrix), int
&widthOFMatrix, int &selectRow) {

       int temp = 0;

       int toLeft = 1;
       int toRight = widthOFMatrix - 1;

       while (toLeft < toRight) {

          for (int i = toRight; i >= toLeft; --i) {

               if (matrix[selectRow][i - 1] > matrix[selectRow][i])
{

                    temp = matrix[selectRow][i];
                    matrix[selectRow][i] = matrix[selectRow][i -
1];
                    matrix[selectRow][i - 1] = temp;

               }

          }
          ++toLeft;

          for (int i = toLeft; i <= toRight; ++i) {

               if (matrix[selectRow][i - 1] > matrix[selectRow][i])
{

                    temp = matrix[selectRow][i];
                    matrix[selectRow][i] = matrix[selectRow][i -
1];
                    matrix[selectRow][i - 1] = temp;

               }
```

```
            }
            --toRight;

        }

    }

    void cocktailSortForMatrix(float **(&matrix), int(&rows),
int(&cols)) {

        system("cls");
        hideCursor(true);
        normalText();

        if (rows == 0) {
            hideCursor(false);
            cout << "В матрице отсутствуют значения!";
            cout << endl << endl;
            system("pause");
            system("cls");
            return;
        }

        int first = 0;
        cout << "Введите первую строку, которую желаете
отсортировать(k):";
        while (!(cin >> first) || (first <= 0) || (first > rows)) {

            clearCinBuff();
            SetConsoleTextAttribute(hStdOut, FOREGROUND_RED |
FOREGROUND_INTENSITY);
            cout << "Ошибка! Введите корректное значение:";

        }
```

```
        SetConsoleTextAttribute(hStdOut, FOREGROUND_GREEN |
FOREGROUND_INTENSITY);

        int second = 0;
        cout << endl << "Введите второю строку, которую желаете
отсортировать(р):";
        while (!(cin >> second) || (second <= 0) || (second > rows))
{

            clearCinBuff();
            SetConsoleTextAttribute(hStdOut, FOREGROUND_RED |
FOREGROUND_INTENSITY);
            cout << "Ошибка! Введите корректное значение:";

        }
        SetConsoleTextAttribute(hStdOut, FOREGROUND_GREEN |
FOREGROUND_INTENSITY);

        --first;
        --second;

        cocktailSortSelectRow(matrix, cols, first);
        cocktailSortSelectRow(matrix, cols, second);

        cout << endl << endl;
        hideCursor(false);
        system("pause");
        system("cls");

    }

        void mainMenu(float **(&matrix), int(&rows), int(&cols)) {

            hideCursor(false);

            string mainMenu[] =
```

```
        {
            "Просмотр состояния матрицы",
            "Инициализация матрицы",
            "Сортировка матрицы",
            "Выход из программы"
        };

        int key = 0;
        int activeMenu = 0;
        short x;
        short y;

        while (true) {

            x = 10, y = 5;

            for (int i = 0; i < size(mainMenu); ++i) {
                if (activeMenu == i) {
                    normalText();
                }
                else {
                    SetConsoleTextAttribute(hStdOut,
FOREGROUND_GREEN);
                }

                gotoxy(x, ++y);
                cout << mainMenu[i];

            }

            key = _getch();
            switch (key) {

            case UP: {
                if (activeMenu > 0) {
                    --activeMenu;
```

```
                    }
                    break;
               }

               case DOWN: {
                    if (activeMenu < size(mainMenu) - 1) {
                         ++activeMenu;
                    }
                    break;
               }

               case ENTER: {
                    switch (activeMenu) {

                    case 0: {
                         outputMatrix(matrix, rows, cols);
                         break;
                    }

                    case 1: {
                         inputMatrix(matrix, rows, cols);
                         break;
                    }

                    case 2: {
                         cocktailSortForMatrix(matrix, rows, cols);
                         break;
                    }

                    case 3: {
                         system("cls");
                         cout << "Завершение работы программы..."
<< endl << endl;
                         system("pause");
                         return;
                    }
```

```
                        break;
                }
                break;
            }

            }

        }

    }

    int main() {

        setlocale(0, "rus");
        SetConsoleTitle(L"Кущевский Андрей, ПЗ-19-1");

        int rows = 0;
        int cols = 0;

        float **matrix = new float *[rows];
        for (int i = 0; i < rows; ++i) {
            matrix[i] = new float[cols];
        }
        normalText();

        mainMenu(matrix, rows, cols);

        for (int i = 0; i < rows; ++i) {
            delete[] matrix[i];
        }

        delete[] matrix;

        _getch();
        return 0;
```

```
}
```

Вікна виконання програми дивитись на рисунках 7.1 – 7.7:

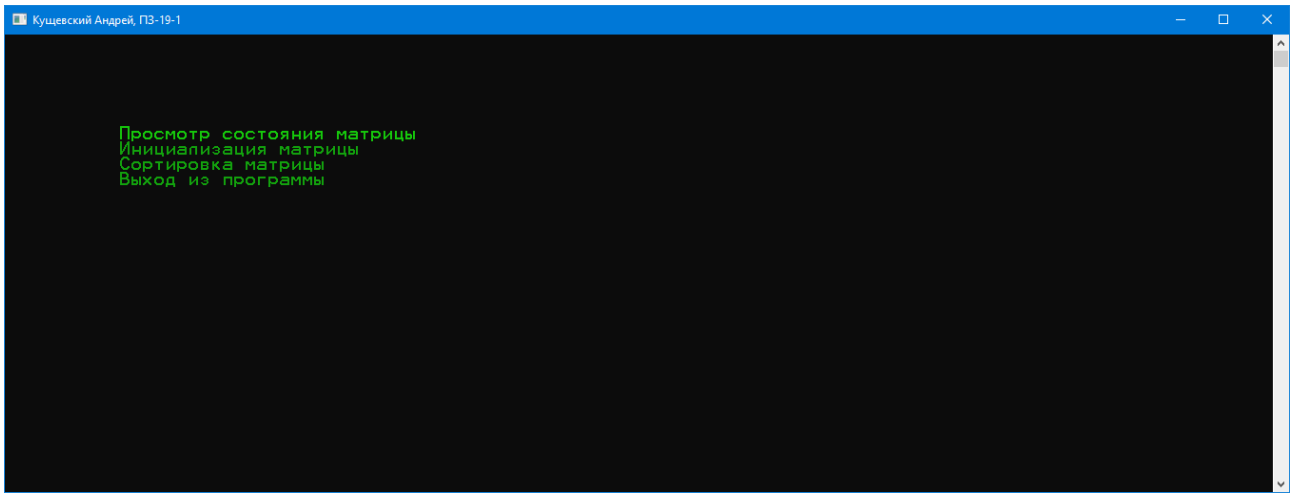


Рисунок 7.1 – Головне меню програми

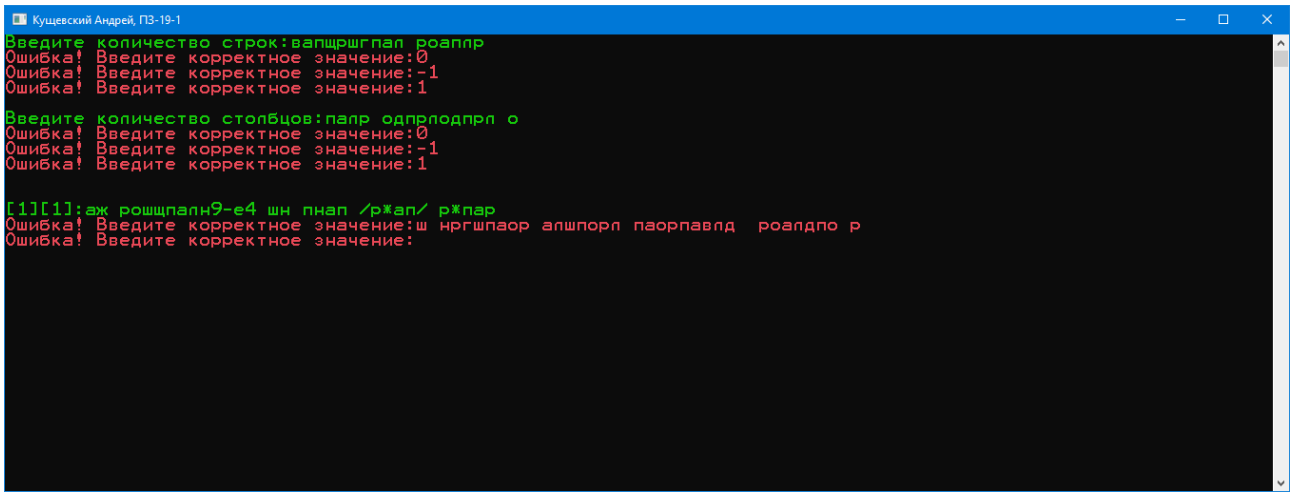


Рисунок 7.2 – Перевірка користувацького друку

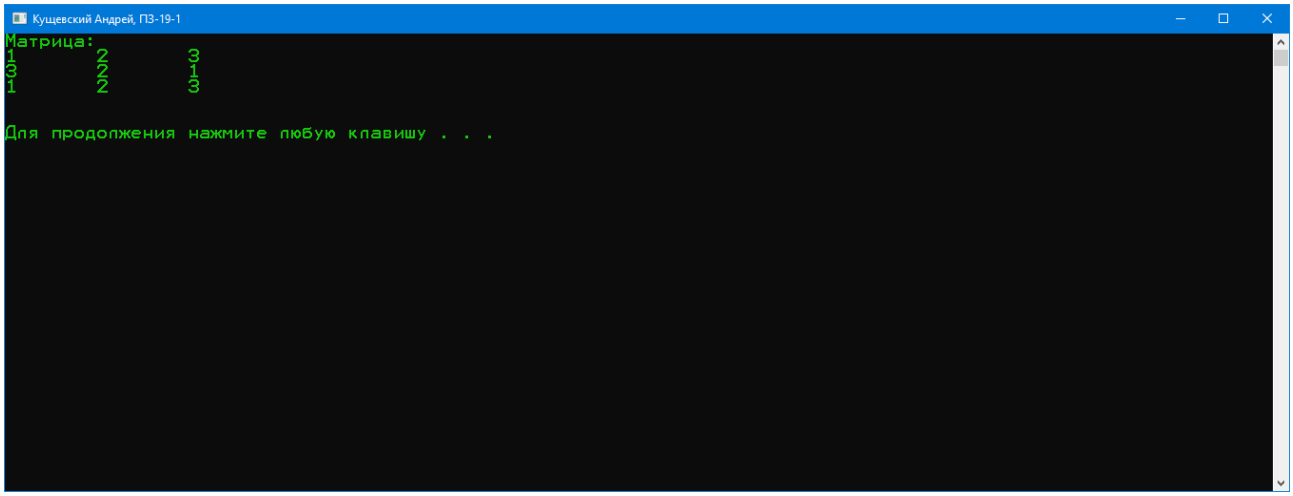


Рисунок 7.3 – Результат роботи функції outputMatrix()

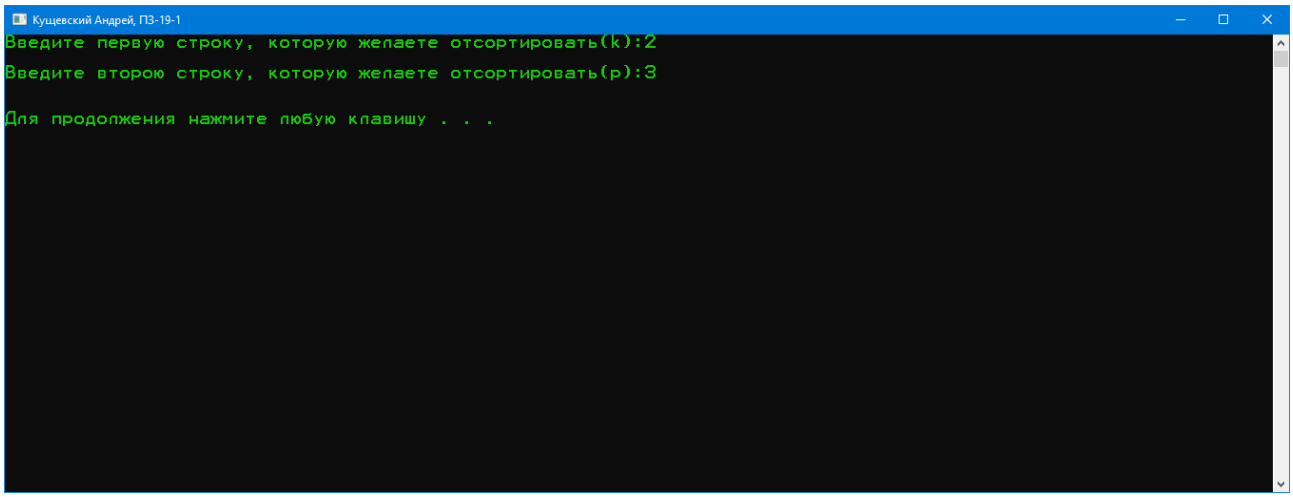


Рисунок 7.4 – Результат роботи функції cocktailSortSelectRow()

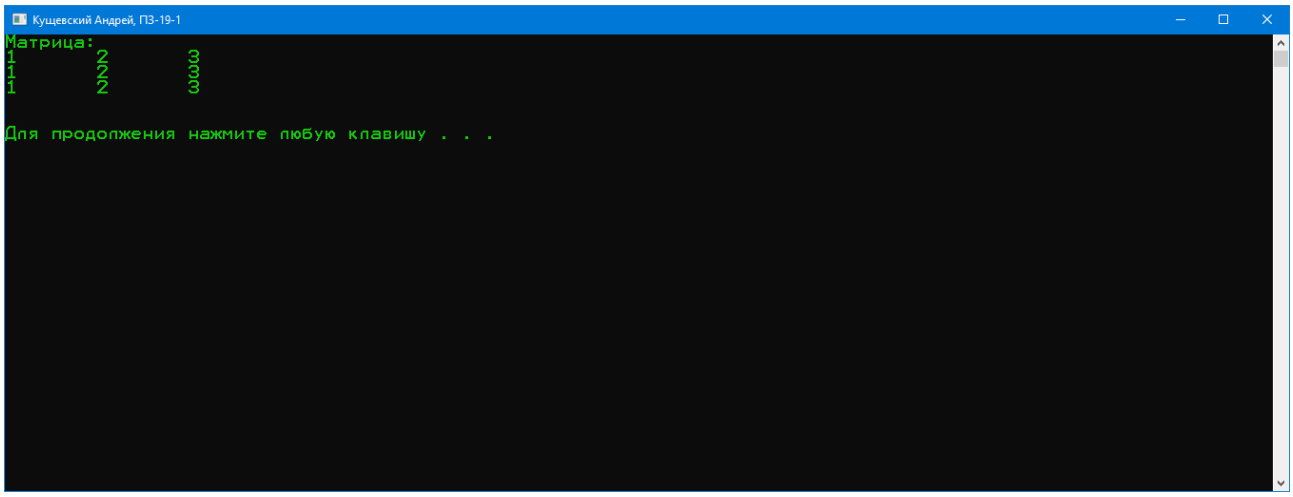


Рисунок 7.5 – Результат роботи функції sortingMatrix()

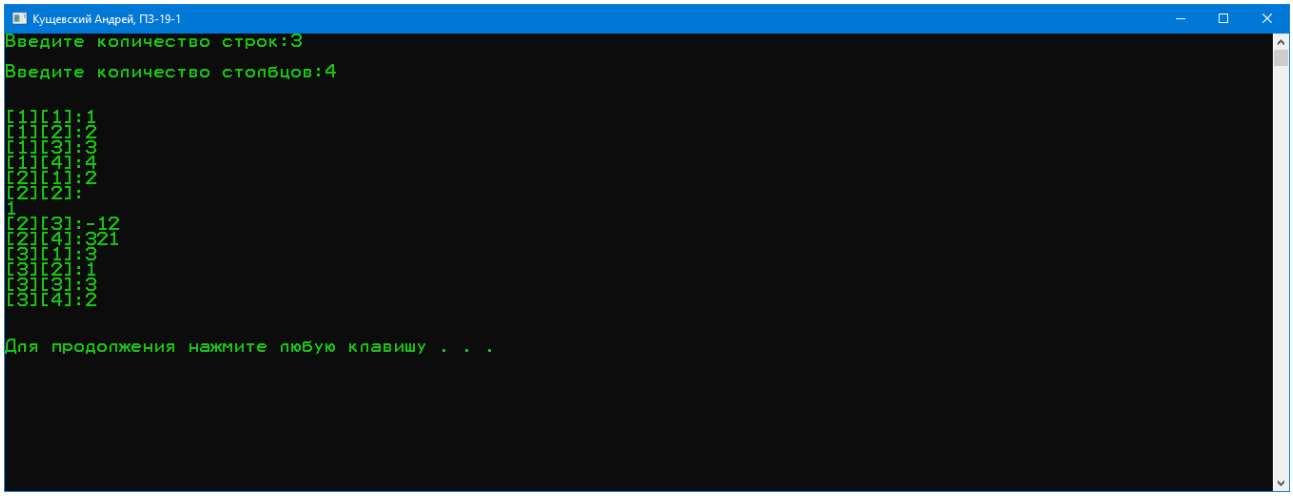


Рисунок 7.6 – Результат роботи функції inputMatrix()

Висновок: отримав навички роботи зі структурованим типом даних масив та реалізації алгоритмів внутрішнього сортування.

ЛАБОРАТОРНА РОБОТА № 9-10

Складання та налагодження програм пошуку в лінійних масивах даних.

Мета: Здобуття навичок реалізації алгоритмів пошуку.

Хід роботи

9.1 Загальна постановка задачі

1 Провести аналіз поставленої задачі.

Загальна постановка завдання:

На основі виконання Лабораторної роботи № 9-10 скласти та налагодити функцію реалізації алгоритму пошукув відсортованих рядках або стовпцях матриці згідно Вашого варіанту. Додати відповідний пункт меню в програму попередньої лабораторної роботи та «підключити» до нього реалізовану функцію пошуку.

2 Розробити та налагодити програму рішення задачі.

3 Оформити звіт з лабораторної роботи.

9.2 Постановка задачі за варіантом

9 Для відсортованих рядків матриці реалізувати алгоритм інтерполяційного пошуку введеного користувачем числа.

2.3 Розробка головної функції

Лістинг 9.1 – Текст програми

```
//Кущевский Андрей, ПЗ-19-1

/*
  Дано двовимірний масив дійсних чисел  a[1..n,1..m] (кількість рядків n та кількість стовпців m вводиться користувачем).
  Впорядкувати за спаданням k-й та p-й рядки масиву використовуючи алгоритм шейкерного сортування (k та  p  вводяться користувачем).
*/
```

```
/*
  Для відсортованих рядків матриці реалізувати алгоритм
інтерполя-ційного пошуку введеного користувачем числа.
*/

#include <iostream>
#include <windows.h>
#include <conio.h>
#include <string>
#include <iomanip>
#include <vector>

#define UP 72
#define DOWN 80
#define ENTER 13

using namespace std;

HANDLE hStdOut = GetStdHandle(STD_OUTPUT_HANDLE);

void hideCursor(bool show) {

  CONSOLE_CURSOR_INFO info;
  info.bVisible = show;
  info.dwSize = 10;
  SetConsoleCursorInfo(hStdOut, &info);

}

void gotoxy(short x, short y) {

  SetConsoleCursorPosition(hStdOut, { x, y });

}
```

```
    void normalText() {

      SetConsoleTextAttribute(hStdOut, FOREGROUND_GREEN |
FOREGROUND_INTENSITY);

    }

    void errorText() {

      SetConsoleTextAttribute(hStdOut, FOREGROUND_RED |
FOREGROUND_INTENSITY);

    }

    void clearCinBuff() {

      cin.clear();
      cin.sync();
      cin.ignore(cin.rdbuf()->in_avail());

    }

    vector<int> indexForInterpolate;

    void sizeOfMatrix(int(&rows), int(&cols)) {

      normalText();
      hideCursor(true);

      cout << "Введите количество строк:";
      while (!(cin >> rows) || (rows <= 0)) {

         clearCinBuff();
         errorText();
         cout << "Ошибка! Введите корректное значение:";
```

```
    }
    normalText();

    cout << endl << "Введите количество столбцов:";
    while (!(cin >> cols) || (cols <= 0)) {

       clearCinBuff();
       errorText();
       cout << "Ошибка! Введите корректное значение:";

    }
    normalText();

}

void outputMatrix(float **matrix, int rows, int cols) {

    system("cls");
    normalText();
    hideCursor(false);

    if (rows == 0) {
       cout << "В матрице отсутствуют значения!";
       cout << endl << endl;
       system("pause");
       system("cls");
       return;
    }

    cout << "Матрица:" << endl;

    for (int i = 0; i < rows; ++i) {
       for (int j = 0; j < cols; ++j) {

            cout << setw(6) << matrix[i][j];
```

```
        }
        cout << endl;
    }

    cout << endl << endl;
    system("pause");
    system("cls");

}

void inputMatrix(float **(&matrix), int(&rows), int(&cols)) {

    system("cls");
    hideCursor(true);
    normalText();

    if (rows != 0) {

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

            delete matrix[i];

        }

        delete[] matrix;

    }

    sizeOfMatrix(rows, cols);

    matrix = new float *[rows];
    for (int i = 0; i < rows; ++i) {

        matrix[i] = new float[cols];

    }
```

```
        cout << endl << endl;

        for (int i = 0; i < rows; ++i) {
           for (int j = 0; j < cols; ++j) {

                cout << "[" << i + 1 << "][" << j + 1 << "]:";
                while (!(cin >> matrix[i][j])) {

                     clearCinBuff();
                     errorText();
                     cout << "Ошибка! Введите корректное значение:";

                }
                normalText();

           }
        }

        cout << endl << endl;
        hideCursor(false);
        system("pause");
        system("cls");

     }

     void cocktailSortSelectRow(float **(&matrix), int
&widthOFMatrix, int &selectRow) {

        float temp = 0;

        int toLeft = 1;
        int toRight = widthOFMatrix - 1;

        while (toLeft <= toRight) {
```

```
          for (int i = toRight; i >= toLeft; --i) {

               if (matrix[selectRow][i - 1] > matrix[selectRow][i])
{

                    temp = matrix[selectRow][i];
                    matrix[selectRow][i] = matrix[selectRow][i -
1];
                    matrix[selectRow][i - 1] = temp;

               }

          }
          ++toLeft;

          for (int i = toLeft; i <= toRight; ++i) {

               if (matrix[selectRow][i - 1] > matrix[selectRow][i])
{

                    temp = matrix[selectRow][i];
                    matrix[selectRow][i] = matrix[selectRow][i -
1];
                    matrix[selectRow][i - 1] = temp;

               }

          }
          --toRight;

       }

     }

     void cocktailSortForMatrix(float **(&matrix), int(&rows),
int(&cols)) {
```

```
        system("cls");
        hideCursor(true);
        normalText();

        if (rows == 0) {
           hideCursor(false);
           cout << "В матрице отсутствуют значения!";
           cout << endl << endl;
           system("pause");
           system("cls");
           return;
        }

        cout << "Исходная матрица:" << endl;
        for (int i = 0; i < rows; ++i) {
           for (int j = 0; j < cols; ++j) {

                cout << setw(6) << matrix[i][j];

           }
           cout << endl;
        }


        int first = 0;
        cout << endl << endl << "Введите первую строку, которую
желаете отсортировать(k):";
        while (!(cin >> first) || (first <= 0) || (first > rows)) {

           clearCinBuff();
           SetConsoleTextAttribute(hStdOut, FOREGROUND_RED |
FOREGROUND_INTENSITY);
           cout << "Ошибка! Введите корректное значение:";

        }
```

```
	SetConsoleTextAttribute(hStdOut, FOREGROUND_GREEN |
FOREGROUND_INTENSITY);

	int second = 0;
	cout << endl << "Введите второю строку, которую желаете
отсортировать(р):";
	while (!(cin >> second) || (second <= 0) || (second > rows))
{

	   clearCinBuff();
	   SetConsoleTextAttribute(hStdOut, FOREGROUND_RED |
FOREGROUND_INTENSITY);
	   cout << "Ошибка! Введите корректное значение:";

	}
	SetConsoleTextAttribute(hStdOut, FOREGROUND_GREEN |
FOREGROUND_INTENSITY);

	--first;
	--second;



	cocktailSortSelectRow(matrix, cols, first);
	cocktailSortSelectRow(matrix, cols, second);

	cout << endl << endl << "Отсортированные строки матрицы:" <<
endl;
	if (first != second) {

	   cout << "[" << first + 1 << "]:";  for (int j = 0; j <
cols; ++j) { cout << setw(5) << matrix[first][j]; };
	   cout << endl;
	   cout << "[" << second + 1 << "]:"; for (int j = 0; j <
cols; ++j) { cout << setw(5) << matrix[second][j]; };

	}
```

```
       else if (first == second) {

          cout << "[" << first + 1 << "]:";  for (int j = 0; j <
cols; ++j) { cout << setw(5) << matrix[first][j]; };

       }

       indexForInterpolate.push_back(first);
       indexForInterpolate.push_back(second);

       cout << endl << endl;
       hideCursor(false);
       system("pause");
       system("cls");

     }

     float interpolateSearchInRow(float **(&matrix), int (&cols),
int (&selectRows), float (&keyOfSearch)) {

       //start intepolateSearch
       float *a = new float[cols];

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

          a[i] = matrix[selectRows][i];

       }

       //объявляем необходимые локальные переменные
       //изначально устанавливаем нижний индекс на начало массива,
       //а верний на конец массива
       int low = 0;
       int high = cols - 1;
       int mid;
       //цикл интерполирующего поиска
```

```
       while (a[low] < keyOfSearch && a[high] >= keyOfSearch)
       {
          //интерполирующий поиск производит оценку новой области
поиска
          //по расстоянию между ключом поиска и текущим значение
элемента
          mid = low + ((keyOfSearch - a[low]) * (high - low)) /
(a[high] - a[low]);
          //если значение в ячейке с индексом mid меньше, то
смещаем нижнюю границу
          if (a[mid] < keyOfSearch)
               low = mid + 1;
          //в случае, если значение больше, то смещаем верхнюю
границу
          else if (a[mid] > keyOfSearch)
               high = mid - 1;
          //если равны, то возвращаем индекс
          else
               return mid;
       }
       //если цикл while не вернул индекс искомого значения,
       //то проверяем не находится ли оно в ячейке массива с
индексом low,
       //иначе возвращаем -1 (значение не найдено)
       if (a[low] == keyOfSearch)
          return low;
       else
          return -1;

       delete[] a;

     }

     void interPolationSearch(float **(&matrix), int (&rows),
int(&cols)) {
```

```
        system("cls");
        hideCursor(true);
        normalText();

        if (cols == 0) {
           hideCursor(false);
           cout << "В матрице отсутствуют значения!";
           cout << endl << endl;
           system("pause");
           system("cls");
           return;
        }

        if (indexForInterpolate.empty()) {
           hideCursor(false);
           cout << "Не было найдено индексов отсортированных строк,
выполните пункт меню \"Сортировка матрицы\"!";
           cout << endl << endl;
           system("pause");
           system("cls");
           return;
        }

        cout << "Матрица:" << endl;

        for (int i = 0; i < rows; ++i) {
           for (int j = 0; j < cols; ++j) {

                cout << setw(6) << matrix[i][j];

           }
           cout << endl;
        }

        int first = 0;
        int second = 0;
```

```
        cout << endl << "Строки, в которых будет совершён
интерполяционный поиск:";
        for (int i = 0; i < indexForInterpolate.size(); ++i) {
           cout << indexForInterpolate[i] + 1 << ' ';
           if (i == 0) {
                first = indexForInterpolate[i];
           }
           else if (i == 1) {
                second = indexForInterpolate[i];
           }
        }
        cout << "!";

        cout << endl << endl << "Введите искомое значение:";
        float findValues = 0;
        while (!(cin >> findValues)) {

           clearCinBuff();
           errorText();
           cout << "Ошибка! Введите корректное значение:";

        }
        normalText();

        int result = 0;

        if (first == second) {
           result = interpolateSearchInRow(matrix, cols, first,
findValues);
           if (result != -1)
                cout << "Значение было найдено в элементе под
индексом [" << first + 1 << "][" << result + 1 << "]!" << endl;
           else
                cout << "Значение не было найдено!" << endl;
```

```
        }
        else if(first != second) {

            result = interpolateSearchInRow(matrix, cols, first,
findValues);
            if (result != -1)
                cout << "Значение было найдено в элементе под
индексом [" << first + 1 << "][" << result + 1 << "]!" << endl;
            else
                cout << "Значение не было найдено!" << endl;

            result = interpolateSearchInRow(matrix, cols, second,
findValues);
            if (result != -1)
                cout << "Значение было найдено в элементе под
индексом [" << first + 1 << "][" << result + 1 << "]!" << endl;
            else
                cout << "Значение не было найдено!" << endl;
        }

        indexForInterpolate.clear();

        cout << endl << endl;
        hideCursor(false);
        system("pause");
        system("cls");

    }

    void mainMenu(float **(&matrix), int(&rows), int(&cols)) {

        hideCursor(false);

        string mainMenu[] =
        {
            "Просмотр состояния матрицы",
```

```
            "Инициализация матрицы",
            "Сортировка матрицы",
            "Интерполяционный поиск",
            "Выход из программы"
        };

        int key = 0;
        int activeMenu = 0;
        short x;
        short y;

        while (true) {

            x = 10, y = 5;

            for (int i = 0; i < size(mainMenu); ++i) {
                if (activeMenu == i) {
                    normalText();
                }
                else {
                    SetConsoleTextAttribute(hStdOut,
FOREGROUND_GREEN);
                }

                gotoxy(x, ++y);
                cout << mainMenu[i];

            }

            key = _getch();
            switch (key) {

            case UP: {
                if (activeMenu > 0) {
                    --activeMenu;
                }
```

```
            break;
        }

        case DOWN: {
            if (activeMenu < size(mainMenu) - 1) {
                ++activeMenu;
            }
            break;
        }

        case ENTER: {
            switch (activeMenu) {

            case 0: {
                outputMatrix(matrix, rows, cols);
                break;
            }

            case 1: {
                inputMatrix(matrix, rows, cols);
                break;
            }

            case 2: {
                cocktailSortForMatrix(matrix, rows, cols);
                break;
            }

            case 3: {
                interPolationSearch(matrix, rows, cols);
                break;
            }

            case 4: {
                system("cls");
```

```
                    cout << "Завершение работы программы..." <<
endl << endl;
                    system("pause");
                    return;
               }

                         break;
               }
               break;
          }

          }

       }

     }

     int main() {

       setlocale(0, "rus");
       SetConsoleTitle(L"Кущевский Андрей, ПЗ-19-1");

       int rows = 0;
       int cols = 0;

       float **matrix = new float *[rows];
       for (int i = 0; i < rows; ++i) {
          matrix[i] = new float[cols];
       }
       normalText();

       mainMenu(matrix, rows, cols);

       for (int i = 0; i < rows; ++i) {
          delete[] matrix[i];
       }
```

```
    delete[] matrix;

    return 0;

}
```

Вікна виконання програми дивитись на рисунках 9.1 – 9.7:

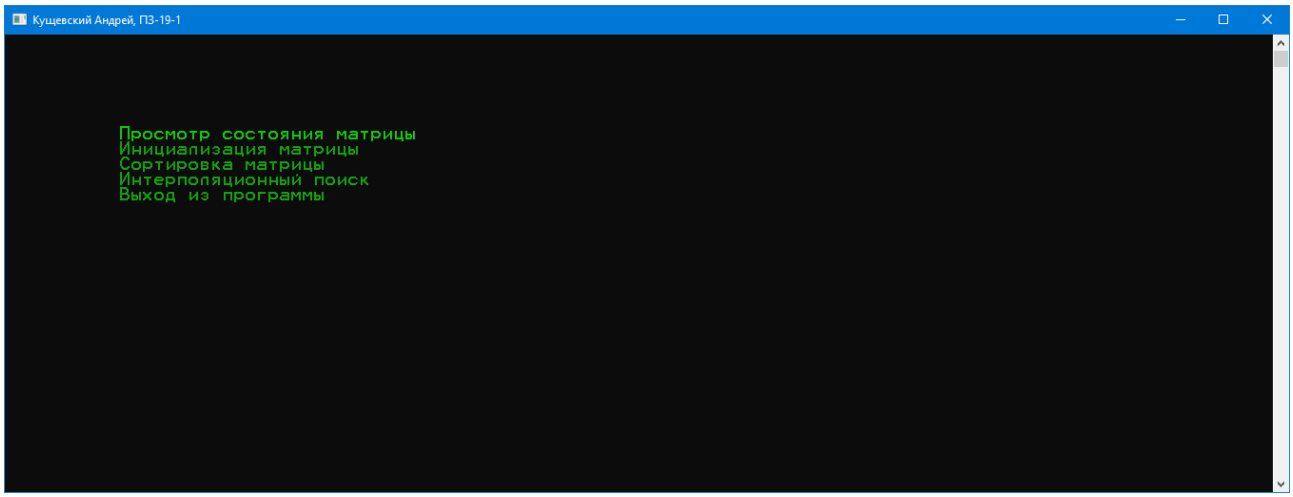


Рисунок 9.1 – Головне меню програми

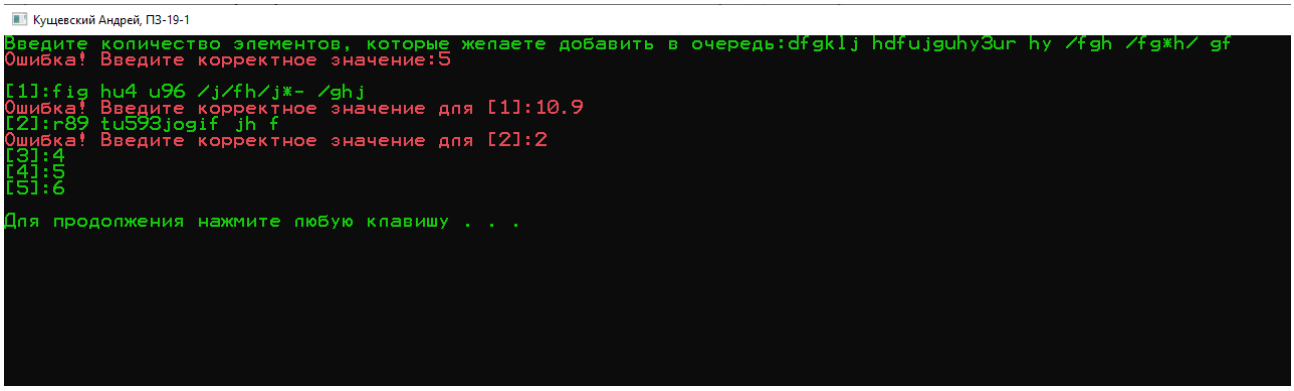


Рисунок 9.2 – Перевірка користувацького друку

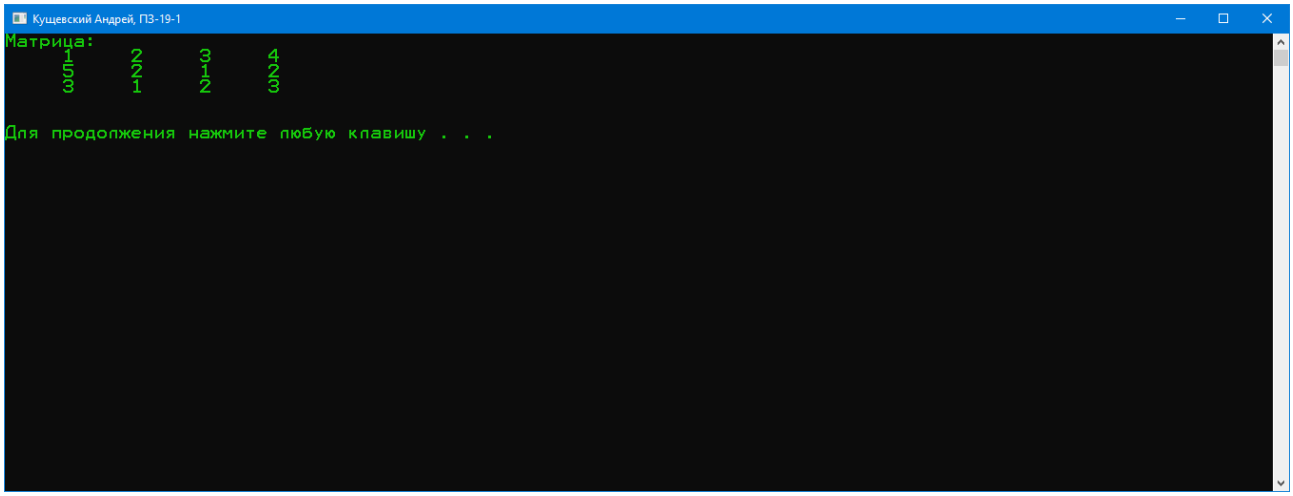


Рисунок 9.3 – Результат роботи функції outputMatrix()

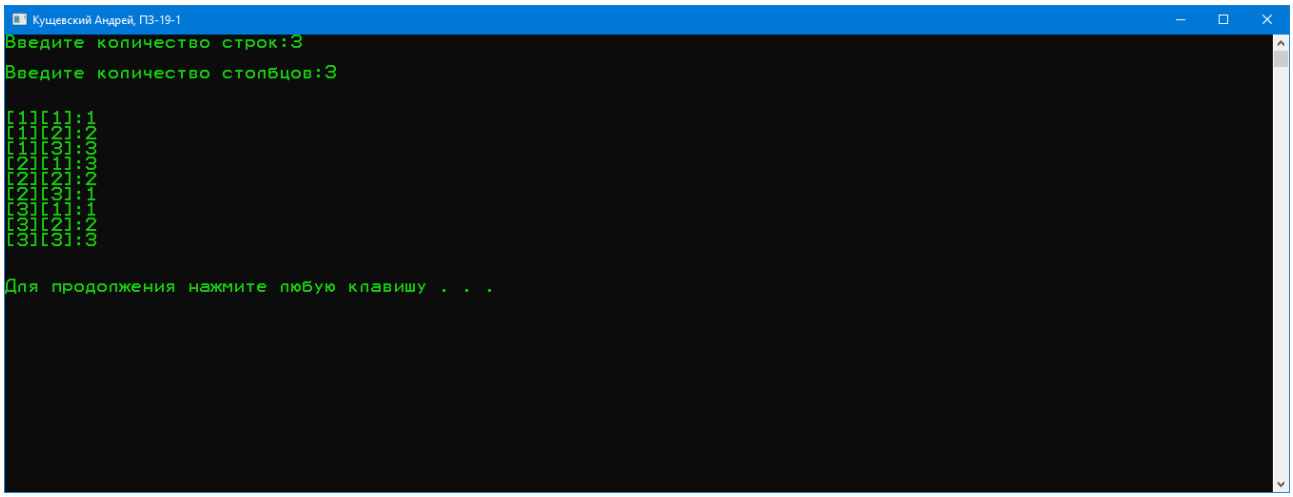


Рисунок 9.4 – Результат роботи функції inputMatrix()

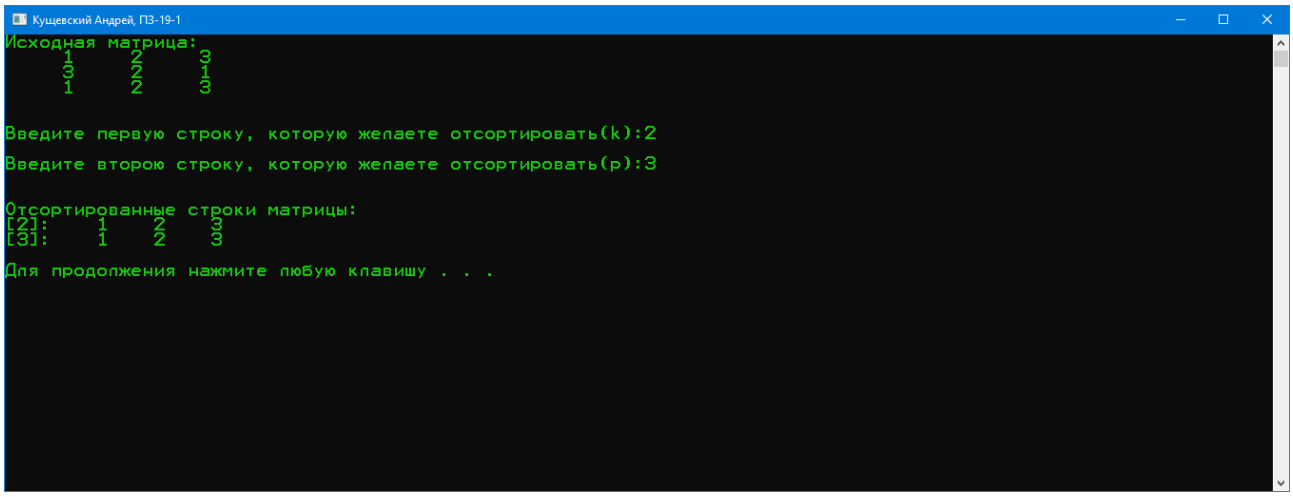


Рисунок 9.5 – Результат роботи функції sortingMatrix()

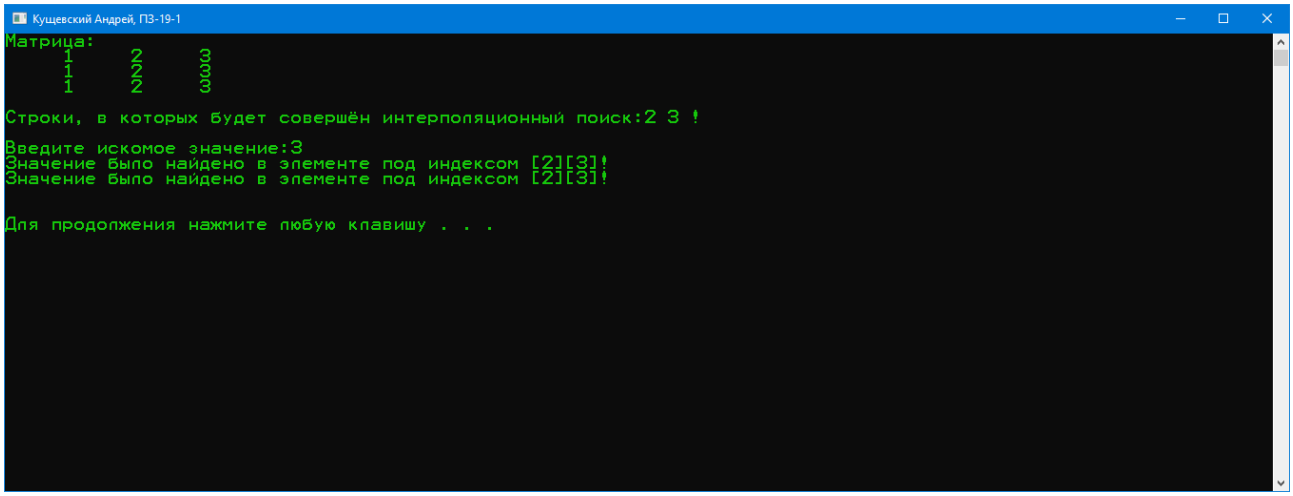


Рисунок 9.6 – Результат роботи функції interpolateSearch()

Висновок: здобуття навичок реалізації алгоритмів пошуку.

ЛАБОРАТОРНА РОБОТА № 11-12

Складання та налагодження програм побудови збалансованого бінарного дерева пошуку та реалізація пошуку ключового елементу.

Мета: отримання навиків в організації динамічної структури даних – збалансованого бінарного дерева пошуку реалізації пошуку ключового елементу.

Хід роботи

11.1 Провести аналіз поставленої задачі.

Загальна постановка завдання:

Скласти та налагодити програму створення збалансованого бінарного дерева на основі нелінійного списку та пошуку ключового елементу. Програма повинна задовольняти наступним вимогам:

- організувати користувацьке меню, яке повинно містити наступні пункти:

1. Створення збалансованого бінарного дерева.
2. Перегляд вмісту бінарного дерева.
3. Пошук ключового елементу.
4. Видалення дерева.

- забезпечити коректне введення користувачем вхідних даних;
- при обробці дерева враховувати, що шукані елементи можуть бути відсутні. В цьому випадку вивести користувачеві відповідне повідомлення;
- введення та виведення вхідних та вихідних даних повинно містити необхідні для користувача повідомлення.

2 Розробити та налагодити програму рішення задачі.

3 Оформити звіт з лабораторної роботи.

Звіт повинен містити наступні розділи:

1 Постановка задачі.

2 Текст програми з відповідними коментарями.

3 Копії вікон виконання програми.

Цей розділ повинен вміщати наступні скріншоти:

– введення невпорядкованої вхідної послідовності даних;

– побудоване збалансоване бінарне дерево пошуку;

– результати пошуку ключового (заданого користувачем) елементу.

3 Висновок.

2.3 Розробка головної функції

Лістинг 11.1 – Текст програми

```
#include <iostream>
#include <Windows.h>
#include <conio.h>
#include <string>
#include <vector>
#include <iomanip>
#define UP 72
#define DOWN 80
#define ENTER 13
using namespace std;


HANDLE hStdOut = GetStdHandle(STD_OUTPUT_HANDLE);

void clearCinBuff()
{

  cin.clear();
  cin.sync();
  cin.ignore(cin.rdbuf()->in_avail());

}
```

```
void endProcedure()
{

  clearCinBuff();
  cout << endl << endl;
  system("pause");
  system("cls");

}

void gotoxy(short x, short y)
{

  SetConsoleCursorPosition(hStdOut, { x, y });

}

void hideCursor(bool showStatus)
{

  _CONSOLE_CURSOR_INFO info;
  info.bVisible = showStatus;
  info.dwSize = 20;
  SetConsoleCursorInfo(hStdOut, &info);

}

void setColorText(short unsigned int item)
{

  SetConsoleTextAttribute(hStdOut, item);

}

//вузли АВЛ-дерева
struct Tree
```

```
{
    float info;
    int h;
    Tree* left;
    Tree* right;
};

 Tree* root = nullptr;

//фукнция для проверки на пустоту дерева
bool isEmpty(Tree* (&Node))
{

  if (!Node)
  {

     return true;

  }
  else
  {

     return false;

  }

}

// три допоміжні функції, пов'язані з висотою height,
bfactor, fixheight

//Перша є обгорткою для поля h, вона може працювати й з
нульовими покажчиками
//(з порожніми деревами). Вона повертає висоту
дерева(піддерева):
int height(Tree* p)
```

```
    {
        if (p != NULL)return p->h;
        else return 0;
    }

    //Друга обчислює balance factor заданого вузла (і працює
тільки з ненульовими покажчиками):
    int bfactor(Tree* p)
    {
        return height(p->right) - height(p->left);
    }

    //Третя функція відновлює коректне значення поля h заданого
вузла
    //(за умови, що значення цього поля в правом і левом дочірніх
вузлах є коректними):
    void fixheight(Tree* p)
    {
        int h1 = height(p->left);
        int h2 = height(p->right);
        if (h1 > h2) p->h = h1 + 1;
        else p->h = h2 + 1;
    }

    //правий простий поворіт
    Tree* rotateright(Tree* p) // правий поворот навколо p
    {
        Tree* q = p->left;
        p->left = q->right;
        q->right = p;
        fixheight(p);
        fixheight(q);
        return q;
    }

    //лівий простий поворіт
```

```
Tree* rotateleft(Tree* q) // лівий поворот навколо q
{
    Tree* p = q->right;
    q->right = p->left;
    p->left = q;
    fixheight(q);
    fixheight(p);
    return p;
}

//балансування вузлу бінарного дерева пошуку
Tree* balance(Tree* p) // балансування вузла p
{
    fixheight(p);
    if (bfactor(p) == 2)
    {
        if (bfactor(p->right) < 0)
            p->right = rotateright(p->right);
        return rotateleft(p);
    }
    if (bfactor(p) == -2)
    {
        if (bfactor(p->left) > 0)
            p->left = rotateleft(p->left);
        return rotateright(p);
    }
    return p;
}

    Tree* create_avltree(Tree*&, float);//функція для створення
бінарного дерева
    void search_tree(Tree*&, float);//функція пошуку ключового
елементу
    void view_tree(Tree*&, int);//функція виводу бінарного дерева
    Tree* exit(Tree*&);//функція видалення бінарного дерева
```

```
    //вставка нового вузла
    Tree* create_avltree(Tree*& p, float k)//функція створення
БД(додавання одного елементу)
    {
        if (p == NULL)
        {
            p = new Tree;
            p->info = k;
            p->h = 0;
            p->left = NULL;
            p->right = NULL;
        }
        else {
            if (k < p->info) p->left = create_avltree(p->left,
k);
            else p->right = create_avltree(p->right, k);
        }
        return balance(p);
    }

    void search_tree(Tree*& tree, float k)//функція пошуку
    {
        if (k == tree->info) { cout << "Елемент найден!"; return;
}
        if (k > tree->info) {
            if (tree->right == NULL) cout << "Елемент не
найден!";
            else search_tree(tree->right, k);
        }
        else {
            if (tree->left == NULL) cout << "Елемент не найден!";
            else search_tree(tree->left, k);
        }
    }

    void view_tree(Tree*& tree, int level)//функція виведення БД
```

```
      //перевірка на існування БД або віток або листків цього БД
     {

         if (tree != NULL) {
             view_tree(tree->right, level + 3);
             if (level != 0)for (int i = 0; i < level + 6; i++)
cout << " ";
             if (level == 0) cout << "Корень:";
             cout << tree->info << endl;
             view_tree(tree->left, level + 3);
         }

     }

     Tree* exit(Tree*& tree)
     {//Видалення усього БД якщо воно існує

         Tree* work = tree;
         if (work->left != NULL) work->left = exit(work->left);
         if (work->right != NULL) work->right = exit(work->right);
         delete work;
         return NULL;

     }

     //очистка выделенной динамической под дерево
     void clearTree(Tree *(&Node))
     {

       //если в дереве отсутствуют ноды, прекращение работы функции
       if (!Node) {
          return;
       }

       clearTree(Node->left);
       clearTree(Node->right);
```

```
        delete Node;

        Node = nullptr;

        return;

    }

    void clearTreeMenu()
    {

        system("cls");
        setColorText(14);

        if (isEmpty(root)) {
            setColorText(12);
            cout << "В дереве отсутствуют элементы, воспользуйтесь
соответствующим пунктом меню для инициализации дерева!";
            endProcedure();
            return;
        }

        cout << "Очистка дерева...";
        clearTree(root);

        endProcedure();
        return;

    }

    void mainMenu()
    {

        hideCursor(false);

        string mainMenu[] =
```

```
{
   "Просмотр состояния дерева",
   "Инициализация дерева значениями",
   "Поиск элемента",
   "Очистка дерева",
   "Выход из программы"
};

short x = 0;
short y = 0;
int key = 0;
int activeMenu = 0;

while (true) {
   hideCursor(false);

   x = 10;
   y = 3;

   for (int i = 0; i < 5; ++i) {

        if (activeMenu == i) {
             setColorText(13);
        }
        else {
             setColorText(14);
        }

        gotoxy(x, ++y);
        cout << mainMenu[i];

   }

   key = getch();

   switch (key) {
```

```
case UP: {
     if (activeMenu > 0) {
          --activeMenu;
     }
     else {
          activeMenu = 5-1;
     }

     break;
}
case DOWN: {

     if (activeMenu < 5-1) {
          ++activeMenu;
     }
     else {
          activeMenu = 0;
     }

     break;
}

case ENTER: {
     switch (activeMenu) {
     case 0: {

          system("cls");
          //Перевірка на існування дерева
          if (root != NULL) {
               view_tree(root, 0);
                    endProcedure();

               system("cls");
          }
          else {
```

```
                              setColorText(12);
                              cout << "В дереве отсутствуют
элементы, воспользуйтесь соответствующим пунктом меню для
инициализации дерева!";
                              endProcedure();
                         }
                         break;
                    }

                    case 1: {
                         system("cls");
          hideCursor(true);

          if (!root) {
             setColorText(12);
             cout << "На данный момент в дереве отсутствуют элементы!"
<< endl << endl;
             setColorText(14);
          }
          else {
             setColorText(12);
             cout << "На данный момент в дереве присутствуют
элементы!" << endl << endl;
             setColorText(14);
          }

          cout << "Введите количество элементов, которые желаете
добавить:";
          int size = 0;
          while (!(cin >> size) || (size <= 0)) {

             setColorText(12);
             clearCinBuff();
             cout << "Ошибка! Введите корректное значение:";

          }
```

```
        setColorText(14);
        cout << endl;

        float value = 0;

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

           cout << "[" << i + 1 << "]:";
           while (!(cin >> value)) {

                setColorText(12);
                clearCinBuff();
                cout << "Ошибка! Введите корректное значение для ["
<< i + 1 << "]:";

           }
           setColorText(14);

           root = create_avltree(root, value);

        }

        hideCursor(false);
        cout << endl;
        system("pause");
        system("cls");
                    break;
                }

                case 2: {
                   if (root != NULL) {
                     system("cls");
                     setColorText(14);
                          float key = 0;
```

```
                        cout << "Введите значение, которе желаете
найти:";
                    cin >> key;
                    search_tree(root, key);
                    endProcedure();
                    }
                    else {
                              setColorText(12);
                              system("cls");
                              cout << "В дереве отсутствуют
элементы, воспользуйтесь соответствующим пунктом меню для
инициализации дерева!";
                              endProcedure();
                    }
                    break;
               }

               case 3: {
                    clearTreeMenu();
                    break;
               }

               case 4: {
                    system("cls");
                    setColorText(14);
                    system("pause");
                    return;
               }

               }
               break;
          }

          }

        }
```

```
}

int main()
{

  SetConsoleCP(1251);
  SetConsoleOutputCP(1251);
  SetConsoleTitle("Кущевский Андрей, ПЗ-19-1");

  mainMenu();
  clearTree(root);

  return 0;

}
```

Вікна виконання програми дивитись на рисунках 11.1 – 11.7:

Рисунок 11.1 – Головне меню програми

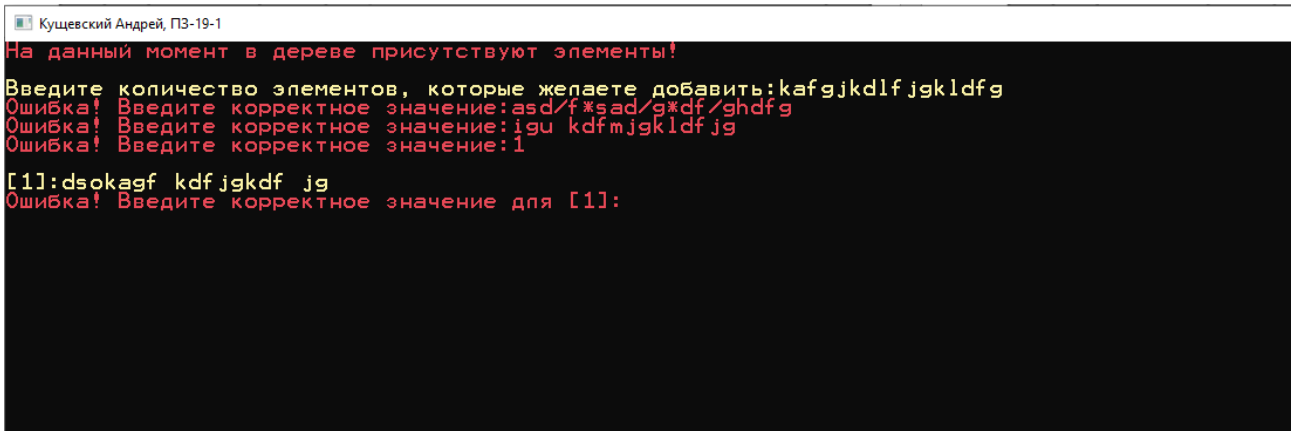


Рисунок 11.2 – Перевірка користувацького друку

Рисунок 11.3 – Результат роботи функції outputTreeMenu()

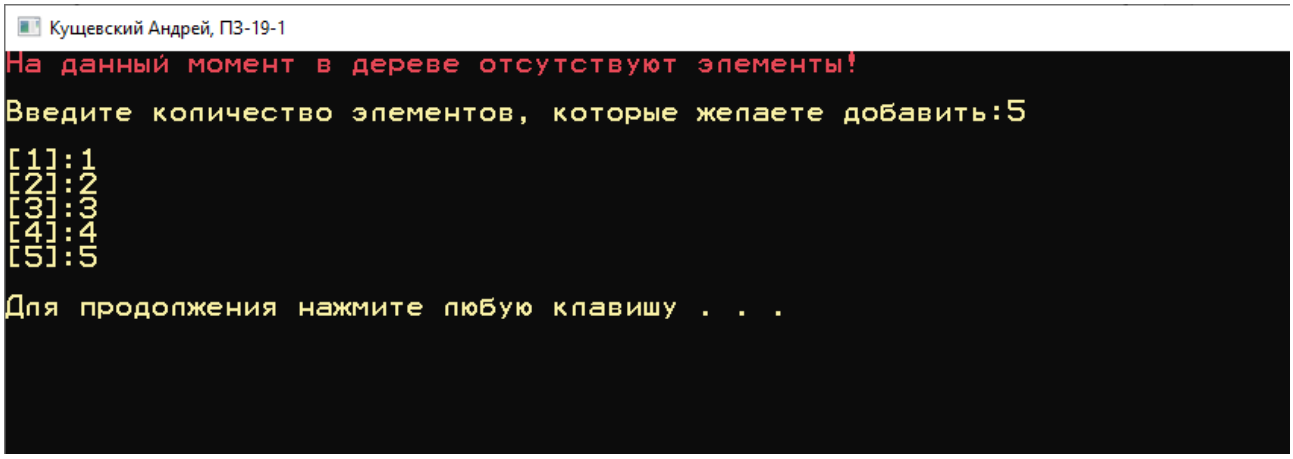


Рисунок 11.4 – Результат роботи функції inputTreeMenu()

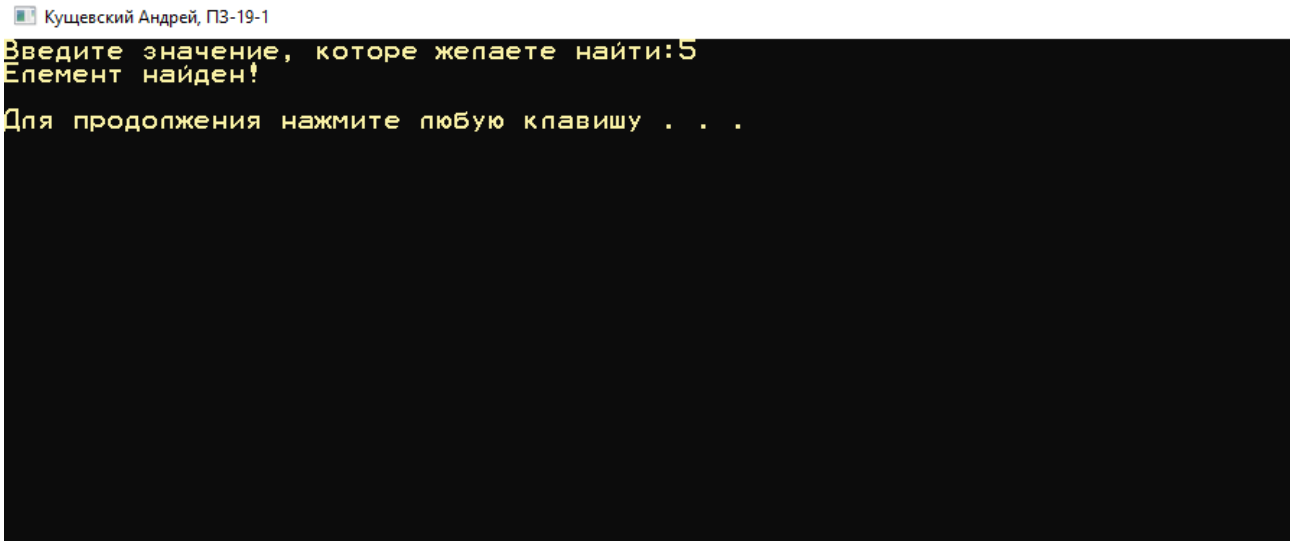


Рисунок 11.5 – Результат роботи функції findTree()

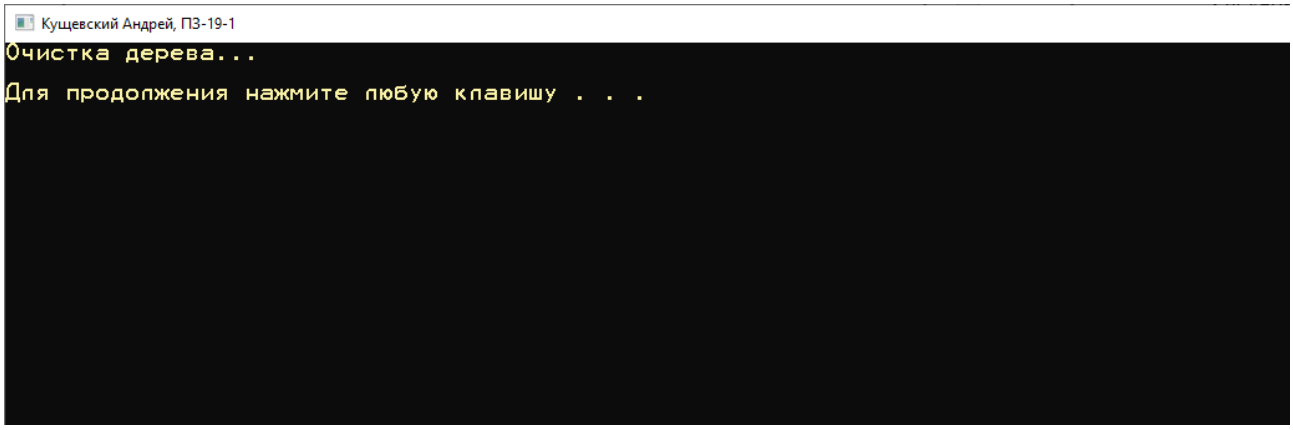


Рисунок 11.6 – Результат роботи функції clearTreeMenu()

Висновок: здобуття навиків в організації динамічної структури даних – збалансованого бінарного дерева пошуку реалізації пошуку ключового елементу.