Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

(МАИ)

**Институт № 8 «Компьютерные науки и прикладная математика»**

**Практические работы**

**по курсу «Алгоритмы и структуры данных»**

**2-й семестр**

**ВАРИАНТ № 22**

|  |  |
| --- | --- |
| **Студент:** | Терентьев М. А. |
| **Группа:** | М8О - 101Б - 23 |
| **Преподаватель:** | Крылов С.С |
| **Подпись:** |  |
| **Оценка:** |  |

Москва, 2024

**Отчет по лабораторной работе №21** по курсу\_\_\_\_\_\_1\_\_\_\_\_\_\_

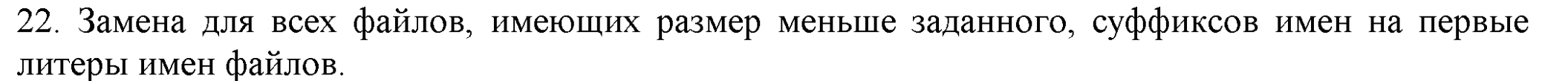
Студент группы: **М8О-101Б-23**, **Терентьев Михаил Андреевич**, № по списку: 22, Контакты**,** [mihateren2003@gmail.com](mailto:mihateren2003@gmail.com), telegram **@mihteren** Работа выполнена: «09» мая 2024г.

Преподаватель: **каф. 806 Крылов Сергей Сергеевич**, Входной контроль знаний с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Отчет сдан « » 2024 г., итоговая оценка \_\_\_\_\_

Подпись преподавателя \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **Тема:** Алгоритмы и структуры данных
2. **Цель работы:** Составить программу выполнения заданных действий над файлами на одном из интерпретируемых командных языков OC UNIX.
3. **Задание:**



1. **Оборудование** (лабораторное):

ЭВМ , процессор , имя узла сети с ОП Мб, НМД Мб. Терминал адрес . Принтер Другие устройства

**Оборудование ПЭВМ студента, если использовалось:**

Процессор **Intel(R) Core(TM) i7-12700H CPU @ 2.70GHz 2.50 GHz** с ОП **16109** Мб, ТТН **476 100** Мб. Встроенный монитор 13 дюймов диагональ, разрешение **2800 \* 1800**.

Другие устройства

1. **Программное обеспечение ЭВМ студента, если использовалось:**

Операционная система семейства **Linux** , наименование **Linux Ubuntu** версия 22.04.01 интерпретатор команд **bash** версия **5.1.16** Система программирования версия Редактор текстов **nano** версия **6.2** Утилиты операционной системы

Прикладные системы и программы

Местонахождение и имена файлов программ и данных на домашнем компьютере

1. **Идея, метод, алгоритм**  решения задачи (в формах: словесной, псевдокода, графической [блок-схема, диаграмма, рисунок, таблица] или формальные спецификации с пред- и постусловиям.

Необходимо создать программу, вычисляющую сумму строк во всех файлах с расширением .h и .cpp, в данной директории и в директориях данной директории.

**Сценарий выполнения работы** [план работы, первоначальный текст программы в черновике (можно на отдельном листе) и тесты либо соображения по тестированию].

аываыва

* min\_size=1000 - Эта строка устанавливает минимальный размер файла в 1000 байт.
* if [ -f "$file" ]; then - Это условие проверяет, является ли текущий элемент файлом (не директорией)
* size=$(wc -c < "$file") - Эта команда использует wc -c для подсчета количества байт в файле, и < "$file" перенаправляет содержимое файла в wc. Результат сохраняется в переменной size.
* if [ $size -lt $min\_size ]; then - Это условие проверяет, меньше ли размер файла min\_size
* new\_name="${file%%.\*}\_$(echo $file | head -c 1)" - Эта строка создает новое имя файла. ${file%%.\*} отсекает расширение файла, head -c 1 берет первый символ имени файла, и это все объединяется с исходным именем файла через символ подчеркивания.
* mv "$file" "$new\_name" - Эта команда переименовывает файл.

min\_size=1000

for file in \*; do

    if [ -f "$file" ]; then

        size=$(wc -c < "$file")

        if [ $size -lt $min\_size ]; then

            new\_name="${file%%.\*}\_$(echo $file | head -c 1)"

            mv "$file" "$new\_name"

            echo "Файл $file переименован в $new\_name"

        fi

    fi

done

1. *Допущен к выполнению работы.*  **Подпись преподавателя \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**
2. **Дневник отладки** должен содержать дату и время сеансов отладки и основные события (ошибки в сценарии и программе, нестандартные ситуации) и краткие комментарии к ним. В дневнике отладки приводятся сведения об использовании других ЭВМ, существенном участии преподавателя и других лиц в написании и отладке программы.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Лаб. или дом. | Дата | Время | Событие | Действие по исправлению | Примечание |
|  |  |  |  |  |  |  |

1. **Замечания автора** по существу работы
2. **Выводы**:. Я составил bash-сценарий, выполняющий указанное действий

Недочёты при выполнении задания могут быть устранены следующим образом:

Подпись студента \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Отчет по лабораторной работе №23** по курсу\_\_\_\_\_\_1\_\_\_\_\_\_\_

Студент группы: **М8О-101Б-23**, **Терентьев Михаил Андреевич**, № по списку: 22, Контакты**,** [mihateren2003@gmail.com](mailto:mihateren2003@gmail.com), telegram **@mihteren** Работа выполнена: «09» мая 2024г.

Преподаватель: **каф. 806 Крылов Сергей Сергеевич**, Входной контроль знаний с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Отчет сдан « » 2024 г., итоговая оценка \_\_\_\_\_

Подпись преподавателя \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **Тема:** Алгоритмы и структуры данных
2. **Цель работы:** Составить программу выполнения преобразований выражений с применением деревьев.
3. **Задание:**



1. **Оборудование** (лабораторное):

ЭВМ , процессор , имя узла сети с ОП Мб, НМД Мб. Терминал адрес . Принтер Другие устройства

**Оборудование ПЭВМ студента, если использовалось:**

Процессор **Intel(R) Core(TM) i7-12700H CPU @ 2.70GHz 2.50 GHz** с ОП **16109** Мб, ТТН **476 100** Мб. Встроенный монитор 13 дюймов диагональ, разрешение **2800 \* 1800**.

Другие устройства

1. **Программное обеспечение ЭВМ студента, если использовалось:**

Операционная система семейства **Linux** , наименование **Linux Ubuntu** версия 22.04.01 интерпретатор команд **bash** версия **5.1.16** Система программирования версия Редактор текстов **nano** версия **6.2** Утилиты операционной системы

Прикладные системы и программы

Местонахождение и имена файлов программ и данных на домашнем компьютере

1. **Идея, метод, алгоритм**  решения задачи (в формах: словесной, псевдокода, графической [блок-схема, диаграмма, рисунок, таблица] или формальные спецификации с пред- и постусловиям.

Необходимо создать структуру данных и функции её обработки. Алгоритм должен будет из данных создавать дерево бинарное дерево. После чего нужно произвести подсчет максимальной ширины дерева.

**Сценарий выполнения работы** [план работы, первоначальный текст программы в черновике (можно на отдельном листе) и тесты либо соображения по тестированию].

Класс TreeNode представляет узел дерева, а класс Tree управляет деревом.

TreeNode:

o value - значение, хранящееся в узле.

o nextSibling - указатель на следующего брата узла.

o firstChild - указатель на первого ребенка узла.

• Tree:

o root - указатель на корень дерева.

Методы класса Tree:

• addNode(int value, int parentValue = -1) - добавляет новый узел в дерево. Если parentValue не указан, узел становится корнем дерева. Если parentValue указан, новый узел становится дочерним для узла с таким значением.

• deleteNode(int value) - удаляет узел и всех его потомков из дерева.

• printTree() - выводит дерево на экран.

• countNodes() - возвращает количество узлов в дереве.

В функции main создается дерево, добавляются узлы, выводится дерево, удаляется узел и выводится обновленное дерево.

Алгоритм работы:

1. Создается пустое дерево.

2. Добавляются узлы в дерево с помощью метода addNode.

3. Выводится дерево на экран с помощью метода printTree.

4. Подсчитывается количество узлов в дереве с помощью метода countNodes.

5. Удаляется узел и его поддерево с помощью метода deleteNode.

6. Выводится обновленное дерево на экран.

7. Подсчитывается количество узлов в обновленном дереве.

8. Деструктор класса Tree автоматически очищает память, выделенную для узлов дерева.

#include <iostream>

using namespace std;

class TreeNode {

public:

    int value;

    TreeNode\* nextSibling;

    TreeNode\* firstChild;

    TreeNode(int val) : value(val), nextSibling(nullptr), firstChild(nullptr) {}

};

class Tree {

private:

    TreeNode\* root;

     int countNodes(TreeNode\* node) const {

        if (!node) return 0;

        int count = 1;

        count += countNodes(node->firstChild);

        count += countNodes(node->nextSibling);

        return count;

    }

    void freeNode(TreeNode\* node) {

        if (!node) return;

        freeNode(node->firstChild);

        freeNode(node->nextSibling);

        delete node;

    }

    TreeNode\* findNode(TreeNode\* node, int value) {

        if (!node) return nullptr;

        if (node->value == value) return node;

        TreeNode\* foundNode = findNode(node->firstChild, value);

        if (foundNode) return foundNode;

        return findNode(node->nextSibling, value);

    }

    TreeNode\* findParent(TreeNode\* node, TreeNode\* child, TreeNode\* parent = nullptr) {

        if (!node) return nullptr;

        if (node->firstChild == child) return node;

        TreeNode\* foundParent = findParent(node->firstChild, child, node);

        if (foundParent) return foundParent;

        return findParent(node->nextSibling, child, parent);

    }

    void removeNodeLinks(TreeNode\* parent, TreeNode\* nodeToDelete) {

        if (!parent || !nodeToDelete) return;

        if (parent->firstChild == nodeToDelete) {

            parent->firstChild = nodeToDelete->nextSibling;

        } else {

            TreeNode\* temp = parent->firstChild;

            while (temp != nullptr && temp->nextSibling != nodeToDelete) {

                temp = temp->nextSibling;

            }

            if (temp != nullptr) {

                temp->nextSibling = nodeToDelete->nextSibling;

            }

        }

        nodeToDelete->nextSibling = nullptr;

        freeNode(nodeToDelete);

    }

    void printTree(TreeNode\* node, int level = 0) const {

        if (node != nullptr) {

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

                cout << "  ";

            }

            cout << node->value << endl;

            printTree(node->firstChild, level + 1);

            printTree(node->nextSibling, level);

        }

    }

public:

    Tree() : root(nullptr) {}

    ~Tree() {

        freeNode(root);

    }

    TreeNode\* addNode(int value, int parentValue = -1) {

        if (parentValue == -1) {

            if (!root) {

                root = new TreeNode(value);

                return root;

            } else {

                cout << "Root already exists." << endl;

                return nullptr;

            }

        } else {

            TreeNode\* parent = findNode(root, parentValue);

            if (parent) {

                TreeNode\* newNode = new TreeNode(value);

                if (!parent->firstChild) {

                    parent->firstChild = newNode;

                } else {

                    TreeNode\* temp = parent->firstChild;

                    while (temp->nextSibling) {

                        temp = temp->nextSibling;

                    }

                    temp->nextSibling = newNode;

                }

                return newNode;

            } else {

                cout << "Parent not found." << endl;

                return nullptr;

            }

        }

    }

    void deleteNode(int value) {

        if (!root) return;

        TreeNode\* nodeToDelete = findNode(root, value);

        if (!nodeToDelete) return;

        if (nodeToDelete == root) {

            freeNode(root);

            root = nullptr;

            return;

        }

        TreeNode\* parent = findParent(root, nodeToDelete);

        removeNodeLinks(parent, nodeToDelete);

    }

    void printTree() {

        printTree(root);

    }

    int countNodes() const {

        return countNodes(root);

    }

};

int main() {

    Tree myTree;

    myTree.addNode(1);

    myTree.addNode(2, 1);

    myTree.addNode(3, 1);

    myTree.addNode(4, 2);

    myTree.addNode(5, 2);

    myTree.addNode(6, 3);

    cout << "Initial tree:" << endl;

    myTree.printTree();

    cout << "Number of nodes in the tree: " << myTree.countNodes() << endl;

    cout << "After deleting node 2 and its subtree:" << endl;

    myTree.deleteNode(2);

    myTree.printTree();

    cout << "Number of nodes in the tree after deleting: " << myTree.countNodes() << endl;

    return 0;

}

}

1. *Допущен к выполнению работы.*  **Подпись преподавателя \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**
2. **Дневник отладки** должен содержать дату и время сеансов отладки и основные события (ошибки в сценарии и программе, нестандартные ситуации) и краткие комментарии к ним. В дневнике отладки приводятся сведения об использовании других ЭВМ, существенном участии преподавателя и других лиц в написании и отладке программы.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Лаб. или дом. | Дата | Время | Событие | Действие по исправлению | Примечание |
|  |  |  |  |  |  |  |

1. **Замечания автора** по существу работы
2. **Выводы**:.Я составил программу выполнения преобразований выражений с применением деревьев

Недочёты при выполнении задания могут быть устранены следующим образом:

Подпись студента \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Отчет по лабораторной работе №24** по курсу\_\_\_\_\_\_1\_\_\_\_\_\_\_

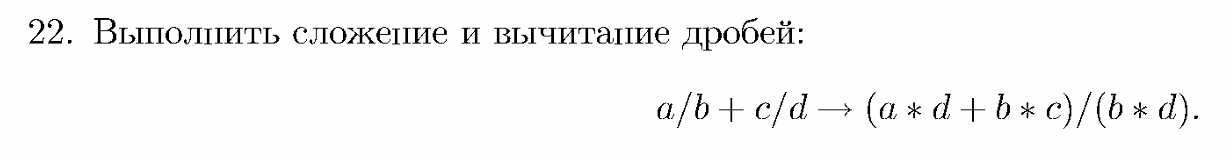
Студент группы: **М8О-101Б-23**, **Терентьев Михаил Андреевич**, № по списку: 22, Контакты**,** [mihateren2003@gmail.com](mailto:mihateren2003@gmail.com), telegram **@mihteren** Работа выполнена: «09» мая 2024г.

Преподаватель: **каф. 806 Крылов Сергей Сергеевич**, Входной контроль знаний с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Отчет сдан « » 2024 г., итоговая оценка \_\_\_\_\_

Подпись преподавателя \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **Тема:** Алгоритмы и структуры данных
2. **Цель работы:** Составить программу выполнения преобразований выражений с применением деревьев.
3. **Задание:**



1. **Оборудование** (лабораторное):

ЭВМ , процессор , имя узла сети с ОП Мб, НМД Мб. Терминал адрес . Принтер Другие устройства

**Оборудование ПЭВМ студента, если использовалось:**

Процессор **Intel(R) Core(TM) i7-12700H CPU @ 2.70GHz 2.50 GHz** с ОП **16109** Мб, ТТН **476 100** Мб. Встроенный монитор 13 дюймов диагональ, разрешение **2800 \* 1800**.

Другие устройства

1. **Программное обеспечение ЭВМ студента, если использовалось:**

Операционная система семейства **Linux** , наименование **Linux Ubuntu** версия 22.04.01 интерпретатор команд **bash** версия **5.1.16** Система программирования версия Редактор текстов **nano** версия **6.2** Утилиты операционной системы

Прикладные системы и программы

Местонахождение и имена файлов программ и данных на домашнем компьютере

1. **Идея, метод, алгоритм**  решения задачи (в формах: словесной, псевдокода, графической [блок-схема, диаграмма, рисунок, таблица] или формальные спецификации с пред- и постусловиям.

Необходимо создать структуру данных и функции её обработки. Алгоритм должен будет из вводимого выражения

создавать дерево выражения, не обязательно с учетом скобок, т.к. произведения. После чего нужно провести преобразование согласно заданию и вывести вид дерева и итоговое выражение

**Сценарий выполнения работы** [план работы, первоначальный текст программы в черновике (можно на отдельном листе) и тесты либо соображения по тестированию].

* Структура данных:

Node: структура для представления узлов дерева, содержит значение (string), указатели на левое и правое поддеревья.

* Построение дерева выражений:

Функции buildExpressionTreeAD, buildExpressionTreeBC, buildExpressionTreeBD: создают поддеревья для частей выражения.

buildExpressionTree: собирает дерево из поддеревьев, представляя выражение вида (ad+bc)/(bd).

* Вывод дерева:

printExpressionTree: рекурсивно выводит дерево в формате, показывающем структуру выражения.

* Обработка ввода:

splitExpression: разделяет входное выражение на две части по символу +.

extractValues: извлекает числовые значения из каждой части выражения.

* Удаление дерева:

deleteExpressionTree: рекурсивно удаляет дерево, освобождая память.

* Основная функция main():

Принимает входное выражение в формате a/b+c/d.

Проверяет корректность ввода и наличие деления на ноль.

Создает и выводит дерево выражения, затем удаляет его.

#include <iostream>

#include <sstream>

#include <string>

using namespace std;

struct Node {

    string value;

    Node\* left;

    Node\* right;

    Node(const string& val) : value(val), left(nullptr), right(nullptr) {}

};

Node\* buildExpressionTreeAD(int a, int d) {

    Node\* root = new Node("\*");

    root->left = new Node(to\_string(a));

    root->right = new Node(to\_string(d));

    return root;

}

Node\* buildExpressionTreeBC(int b, int c) {

    Node\* root = new Node("\*");

    root->left = new Node(to\_string(b));

    root->right = new Node(to\_string(c));

    return root;

}

Node\* buildExpressionTreeBD(int b, int d) {

    Node\* root = new Node("\*");

    root->left = new Node(to\_string(b));

    root->right = new Node(to\_string(d));

    return root;

}

Node\* buildExpressionTree(int a, int b, int c, int d) {

    Node\* ad = buildExpressionTreeAD(a, d);

    Node\* bc = buildExpressionTreeBC(b, c);

    Node\* bd = buildExpressionTreeBD(b, d);

    Node\* adPlusBc = new Node("+");

    adPlusBc->left = ad;

    adPlusBc->right = bc;

    Node\* root = new Node("/");

    root->left = adPlusBc;

    root->right = bd;

    return root;

}

void printExpressionTree(const Node\* node, int level = 0) {

    if (node == nullptr) return;

    printExpressionTree(node->right, level + 1);

    for (int i = 0; i < level; ++i) {

        cout << "   ";

    }

    cout << node->value << endl;

    printExpressionTree(node->left, level + 1);

}

void deleteExpressionTree(Node\* node) {

    if (node == nullptr) return;

    deleteExpressionTree(node->left);

    deleteExpressionTree(node->right);

    delete node;

}

bool splitExpression(const string& expression, string& part1, string& part2) {

    size\_t plusPos = expression.find('+');

    if (plusPos == string::npos) return false;

    part1 = expression.substr(0, plusPos);

    part2 = expression.substr(plusPos + 1);

    return true;

}

bool extractValues(const string& part1, const string& part2, int& a, int& b, int& c, int& d) {

    istringstream iss1(part1), iss2(part2);

    char div1, div2;

    iss1 >> a >> div1 >> b;

    iss2 >> c >> div2 >> d;

    return div1 == '/' && div2 == '/';

}

int main() {

    string expression;

    cout << "Enter expr in format a/b+c/d: ";

    getline(cin, expression);

    string part1, part2;

    if (!splitExpression(expression, part1, part2)) {

        cerr << "Error: incorrect input format." << endl;

        return 1;

    }

    int a, b, c, d;

    if (!extractValues(part1, part2, a, b, c, d)) {

        cerr << "Error: incorrect input format." << endl;

        return 1;

    }

    if (b == 0 || d == 0) {

        cerr << "Error: division by zero is not allowed." << endl;

        return 1;

    }

    Node\* root = buildExpressionTree(a, b, c, d);

    cout << "Expr tree for (ad+bc)/(bd):" << endl;

    printExpressionTree(root);

    deleteExpressionTree(root);

    return 0;

}

1. *Допущен к выполнению работы.*  **Подпись преподавателя \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**
2. **Дневник отладки** должен содержать дату и время сеансов отладки и основные события (ошибки в сценарии и программе, нестандартные ситуации) и краткие комментарии к ним. В дневнике отладки приводятся сведения об использовании других ЭВМ, существенном участии преподавателя и других лиц в написании и отладке программы.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Лаб. или дом. | Дата | Время | Событие | Действие по исправлению | Примечание |
|  |  |  |  |  |  |  |

1. **Замечания автора** по существу работы
2. **Выводы**:. Я составил программу выполнения преобразований выражений с применением деревьев

Недочёты при выполнении задания могут быть устранены следующим образом:

Подпись студента \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Отчет по лабораторной работе №26** по курсу\_\_\_\_\_\_1\_\_\_\_\_\_\_

Студент группы: **М8О-101Б-23**, **Терентьев Михаил Андреевич**, № по списку: 22, Контакты**,** [mihateren2003@gmail.com](mailto:mihateren2003@gmail.com), telegram **@mihteren** Работа выполнена: «09» мая 2024г.

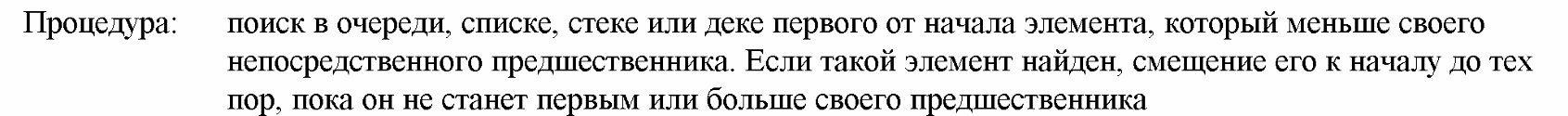
Преподаватель: **каф. 806 Крылов Сергей Сергеевич**, Входной контроль знаний с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Отчет сдан « » 2024 г., итоговая оценка \_\_\_\_\_

Подпись преподавателя \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **Тема:** Алгоритмы и структуры данных
2. **Цель работы:** Составить и отладить модуль реализации по заданной схеме модуля определений для абстрактного (пользовательского) типа данных (стека, очереди, дека или списка). Составить программный модуль, сортирующий экземпляр указанного абстрактного типа данных заданным методом.
3. **Задание:**





1. **Оборудование** (лабораторное):

ЭВМ , процессор , имя узла сети с ОП Мб, НМД Мб. Терминал адрес . Принтер Другие устройства

**Оборудование ПЭВМ студента, если использовалось:**

Процессор **Intel(R) Core(TM) i7-12700H CPU @ 2.70GHz 2.50 GHz** с ОП **16109** Мб, ТТН **476 100** Мб. Встроенный монитор 13 дюймов диагональ, разрешение **2800 \* 1800**.

Другие устройства

1. **Программное обеспечение ЭВМ студента, если использовалось:**

Операционная система семейства **Linux** , наименование **Linux Ubuntu** версия 22.04.01 интерпретатор команд **bash** версия **5.1.16** Система программирования версия Редактор текстов **nano** версия **6.2** Утилиты операционной системы

Прикладные системы и программы

Местонахождение и имена файлов программ и данных на домашнем компьютере

1. **Идея, метод, алгоритм**  решения задачи (в формах: словесной, псевдокода, графической [блок-схема, диаграмма, рисунок, таблица] или формальные спецификации с пред- и постусловиям.

Необходимо создать структуру данных и функции её обработки. Алгоритм должен из данных создавать 2 стэка и производить их конкатенаци. После чего, с помощью реализации быстрой сортировки и реализации стэка необходимо отсортировать заданный массив.

**Сценарий выполнения работы** [план работы, первоначальный текст программы в черновике (можно на отдельном листе) и тесты либо соображения по тестированию].

1. • **Структура данных**:
   * Node: структура для узлов списка, содержит данные (data\_type) и указатель на следующий узел (next).
   * List: структура для списка, содержит указатели на голову (head) и хвост (tail) списка, а также размер (size).
2. **Операции со списком**:
   * list\_create(): создает новый пустой список.
   * list\_is\_empty(): проверяет, пуст ли список.
   * list\_push\_front(), list\_push\_back(): добавляют элементы в начало и конец списка соответственно.
   * list\_pop\_front(), list\_pop\_back(): удаляют элементы из начала и конца списка соответственно.
   * list\_print(): выводит элементы списка.
   * list\_size(): возвращает размер списка.
   * list\_insert(): вставляет элемент в конец списка.
   * list\_erase(): удаляет элемент из списка по значению ключа.

#include <iostream>

using namespace std;

typedef int data\_type;

typedef int key\_type;

struct Node {

    data\_type data;

    Node\* next;

};

struct List {

    Node\* head;

    Node\* tail;

    int size;

};

List\* list\_create() {

    List\* list = new List;

    list->head = nullptr;

    list->tail = nullptr;

    list->size = 0;

    return list;

}

bool list\_is\_empty(const List\* list) {

    return list->size == 0;

}

void list\_push\_front(List\* list, const data\_type& value) {

    Node\* newNode = new Node;

    newNode->data = value;

    newNode->next = list->head;

    list->head = newNode;

    if (list->tail == nullptr) {

        list->tail = newNode;

    }

    list->size++;

}

void list\_push\_back(List\* list, const data\_type& value) {

    Node\* newNode = new Node;

    newNode->data = value;

    newNode->next = nullptr;

    if (list->tail != nullptr) {

        list->tail->next = newNode;

    } else {

        list->head = newNode;

    }

    list->tail = newNode;

    list->size++;

}

void list\_pop\_front(List\* list) {

    if (!list\_is\_empty(list)) {

        Node\* temp = list->head;

        list->head = list->head->next;

        delete temp;

        list->size--;

        if (list->head == nullptr) {

            list->tail = nullptr;

        }

    }

}

void list\_pop\_back(List\* list) {

    if (!list\_is\_empty(list)) {

        if (list->head == list->tail) {

            delete list->head;

            list->head = list->tail = nullptr;

        } else {

            Node\* current = list->head;

            while (current->next != list->tail) {

                current = current->next;

            }

            delete list->tail;

            list->tail = current;

            list->tail->next = nullptr;

        }

        list->size--;

    }

}

void list\_print(const List\* list) {

    Node\* current = list->head;

    while (current != nullptr) {

        cout << current->data << " ";

        current = current->next;

    }

    cout << endl;

}

int list\_size(const List\* list) {

    return list->size;

}

void list\_insert(List\* list, const data\_type& value) {

    Node\* newNode = new Node;

    newNode->data = value;

    newNode->next = nullptr;

    if (list->head == nullptr) {

        list->head = list->tail = newNode;

    } else {

        list->tail->next = newNode;

        list->tail = newNode;

    }

    list->size++;

}

void list\_erase(List\* list, const key\_type& key) {

    if (!list\_is\_empty(list)) {

        Node\* current = list->head;

        Node\* prev = nullptr;

        while (current != nullptr) {

            if (current->data == key) {

                if (prev != nullptr) {

                    prev->next = current->next;

                } else {

                    list->head = current->next;

                }

                delete current;

                list->size--;

                return;

            }

            prev = current;

            current = current->next;

        }

    }

}

void task(List\* list) {

    if (list->head == nullptr || list->head == list->tail) {

        return;

    }

    Node\* prev = list->head;

    Node\* current = list->head->next;

    while (current != nullptr) {

        if (current->data < prev->data) {

            prev->next = current->next;

            if (current == list->tail) {

                list->tail = prev;

            }

            current->next = list->head;

            list->head = current;

            return;

        }

        prev = current;

        current = current->next;

    }

}

int main() {

    List\* myList = list\_create();

    list\_insert(myList, 5);

    list\_insert(myList, 2);

    list\_insert(myList, 7);

    list\_insert(myList, 1);

    list\_insert(myList, 9);

    cout << "Original list: ";

    list\_print(myList);

    task(myList);

    cout << "List after moving smaller element to front: ";

    list\_print(myList);

    // Освобождаем память

    while (!list\_is\_empty(myList)) {

        list\_pop\_front(myList);

    }

    delete myList;

    return 0;

}

}

1. *Допущен к выполнению работы.*  **Подпись преподавателя \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**
2. **Дневник отладки** должен содержать дату и время сеансов отладки и основные события (ошибки в сценарии и программе, нестандартные ситуации) и краткие комментарии к ним. В дневнике отладки приводятся сведения об использовании других ЭВМ, существенном участии преподавателя и других лиц в написании и отладке программы.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Лаб. или дом. | Дата | Время | Событие | Действие по исправлению | Примечание |
|  |  |  |  |  |  |  |

1. **Замечания автора** по существу работы
2. **Выводы**:. Я составил программу выполнения заданного преобразования с использованием структуры стэка, которую сам и реализовал, также поработал над сортировкой

Недочёты при выполнении задания могут быть устранены следующим образом:

Подпись студента \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Отчёт по практикуму по циклу дисциплин «Информатика».**

**Задание №6.**

Студент группы: **М8О-101Б-23**, **Терентьев Михаил Андреевич**, № по списку: 22, Контакты**,** [mihateren2003@gmail.com](mailto:mihateren2003@gmail.com), telegram **@mihteren** Работа выполнена: «09» мая 2024г.

Преподаватель: **каф. 805 Крылов Сергей Сергеевич**

Входной контроль знаний с оценкой: **\_\_\_\_\_**

Отчет сдан « » 2024 г.

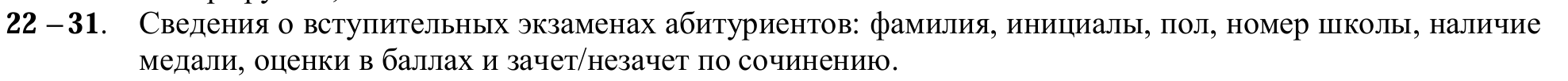
Итоговая оценка: \_\_\_\_\_

Подпись преподавателя: \_\_\_\_\_\_\_\_\_\_

1. **Тема:** Разработать последовательную структуру данных для представления простейшей базы данных на

файлах в СП Си в соответствии с заданным вариантом.

2. **Цель работы:** разработать последовательную структуру данных для представления простейшей базы данных на файлах в СП Си в соответствии с заданным вариантом.

3. **Задание: **

4. **Оборудование**: *Оборудование ПЭВМ студента, если использовалось:*

Процессор **Intel(R) Core(TM) i7-12700H CPU @ 2.70GHz 2.50 GHz** с ОП **16109** Мб, ТТН **476 100** Мб. Встроенный монитор 13 дюймов диагональ, разрешение **2800 \* 1800**.

5. **Программное обеспечение:** *Программное обеспечение ПЭВМ студента, если использовалось****:***

Операционная система семейства **Ubuntu**, наименование версия **VirtualBox Ubuntu 20.04.3 LTS**,

интерпретатор команд **bash** версия **5.0.17**. Система программирования **C**.Редактор текстов **VI** версия **8.1**

6**. Идея, метод, алгоритм** *решения задачи [в формах: словесной, псевдокода, графической (блок-схема, диаграмма, рисунок, таблица) или формальные спецификации с пред- и постусловиями]:*

Задача состоит в том, чтобы обработать данные из заданного текстового файла. Результатом должно быть выведение информации о группе и информации студентки, с максимальным средним баллом.

7. **Сценарий выполнения работы** *(план работы, первоначальный текст программы в черновике (можно на отдельном листе) и тесты либо соображения по тестированию)*.

Сценарий выполнения работы:

1. Подготовить тестовые файлы, с помощью которых будет показана корректность работы разработанной программы.
2. Придумать алгоритм решения поставленной задачи.
3. Используя функции заданной системы программирования, реализовать придуманный алгоритм в виде трёх программ.
4. Провести тестирование программы, чтобы убедиться в корректности её работы.
5. Сформировать протокол.

Applicants\_dump.cpp

#include <iostream>

#include <fstream>

#include "person.h"

const char\* lastNames[] = {"Petrov", "Ivanov", "Sidorov", "Kotov", "Agapov"};

const char\* firstNames[] = {"Ivan", "Petr", "Oleg", "Alex", "Lev"};

void strcpy(char\* dest, const char\* src) {

    while (\*src) {

        \*dest++ = \*src++;

    }

    \*dest = '\0';

}

int randRange(int min, int max) {

    return rand() % (max - min + 1) + min;

}

void generateApplicants(const char\* filename, int count) {

    std::ofstream file(filename, std::ios::binary);

    if (!file) {

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

        return;

    }

    for (int i = 0; i < count; ++i) {

        Applicant applicant;

        int lastNameIndex = randRange(0, 4);

        strcpy(applicant.lastName, lastNames[lastNameIndex]);

        int firstNameIndex = randRange(0, 4);

        strcpy(applicant.initials, firstNames[firstNameIndex]);

        applicant.initials[1] = '.';

        applicant.initials[2] = '\0';

        applicant.gender = randRange(0, 1) == 0 ? 'M' : 'F';

        applicant.schoolNumber = randRange(1, 100);

        applicant.medal = randRange(0, 1) == 0;

        applicant.score = randRange(0, 100);

        applicant.pass = applicant.score >= 50;

        file.write(reinterpret\_cast<const char\*>(&applicant), sizeof(Applicant));

    }

    file.close();

}

int main(int argc, char\* argv[]) {

    if (argc != 3) {

        std::cerr << "Usage: " << argv[0] << " filename count" << std::endl;

        return 1;

    }

    const char\* filename = argv[1];

    int count = atoi(argv[2]);

    if (count <= 0) {

        std::cerr << "Count must be a positive integer" << std::endl;

        return 1;

    }

    generateApplicants(filename, count);

    std::cout << "Generated " << count << " applicants to " << filename << std::endl;

    return 0;

}

Medalist\_below\_scope.cpp

#include <iostream>

#include <fstream>

#include "person.h"

#include <iomanip>

using namespace std;

int atoi(const char\* str) {

    int res = 0;

    for (int i = 0; str[i] != '\0'; ++i) {

        if (str[i] < '0' || str[i] > '9') {

            return -1;

        }

        res = res \* 10 + str[i] - '0';

    }

    return res;

}

void printMedalistBelowScore(const char\* filename, int score) {

    ifstream file(filename, ios::binary);

    if (!file) {

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

        return;

    }

    Applicant applicant;

    while (file.read(reinterpret\_cast<char\*>(&applicant), sizeof(Applicant))) {

        if (applicant.medal && applicant.score < score) {

            cout << applicant.lastName << " " << applicant.initials << endl;

        }

    }

    file.close();

}

void printApplicants(const char\* filename) {

    ifstream file(filename, ios::binary);

    if (!file) {

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

        return;

    }

    Applicant applicant;

    cout << left << setw(15) << "Last Name"

         << setw(15) << "Initials"

         << setw(10) << "Gender"

         << setw(15) << "School Number"

         << setw(10) << "Medal"

         << setw(10) << "Score"

         << setw(10) << "Pass" << endl;

    cout << "-----------------------------------------------------------------------------" << endl;

    while (file.read(reinterpret\_cast<char\*>(&applicant), sizeof(Applicant))) {

        cout << left << setw(15) << applicant.lastName

             << setw(15) << applicant.initials

             << setw(10) << applicant.gender

             << setw(15) << applicant.schoolNumber

             << setw(10) << (applicant.medal ? "Yes" : "No")

             << setw(10) << applicant.score

             << setw(10) << (applicant.pass ? "Yes" : "No") << endl;

    }

    file.close();

}

int main(int argc, char\* argv[]) {

    if (argc < 2 || argc > 3) {

        cerr << "Usage: " << argv[0] << " filename [score]" << endl;

        return 1;

    }

    const char\* filename = argv[1];

    if (argc == 2) {

        printApplicants(filename);

    } else if (argc == 3) {

        int score = atoi(argv[2]);

        if (score < 0 || score > 100) {

            cerr << "Score must be between 0 and 100" << endl;

            return 1;

        }

        printMedalistBelowScore(filename, score);

    }

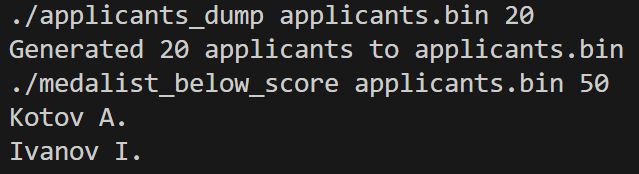
    return 0;

}

*Пункты 1-7 отчета составляются строго до начала лабораторной работы.*

*Допущен к выполнению работы.*  **Подпись преподавателя \_\_\_\_\_\_\_**

8. **Распечатка протокола** *(подклеить листинг окончательного варианта программы с тестовыми примерами, подписанный преподавателем).*



-----------------------------------------------------------------------------

9. **Дневник отладки** должен содержать дату и время сеансов отладки и основные события (ошибки в сценарии и программе, нестандартные ситуации) и краткие комментарии к ним. В дневнике отладки приводятся сведения об использовании других ЭВМ, существенном участии преподавателя и других лиц в написании и отладке программы.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Лаб. или дом.** | **Дата** | **Время** | **Событие** | **Действие по исправлению** | **Примечание** |
|  |  |  |  |  |  |  |

10. **Замечания автора** по существу работы: замечания отсутствуют.

11. **Вывод:** в ходе выполнения данного задания практикума я научился работать с файлами с СП Си.

Подпись студента \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Отчёт по практикуму по циклу дисциплин «Информатика».**

**Задание №8.**

Студент группы: **М8О-101Б-23**, **Терентьев Михаил Андреевич**, № по списку: 22, Контакты**,** [mihateren2003@gmail.com](mailto:mihateren2003@gmail.com), telegram **@mihteren** Работа выполнена: «09» мая 2024г.

Преподаватель: **каф. 805 Крылов Сергей Сергеевич**

Входной контроль знаний с оценкой: **\_\_\_\_\_**

Отчет сдан « » 2024 г.

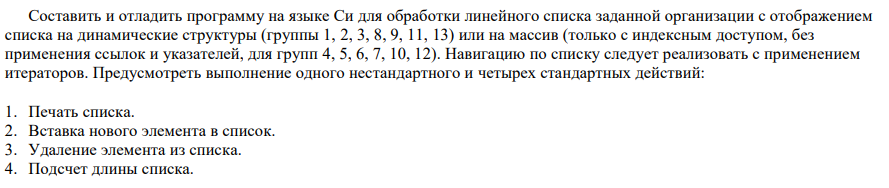
Итоговая оценка: \_\_\_\_\_

Подпись преподавателя: \_\_\_\_\_\_\_\_\_\_

1. **Тема:** Линейные списки.

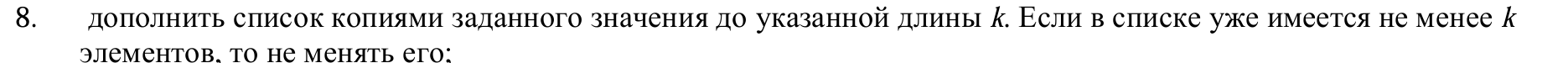
2. **Цель работы:** Составить и отладить программу на языке Си для обработки линейного списка заданной

организации с отображением списка на динамические структуры.

3. **Задание: **







4. **Оборудование**: *Оборудование ПЭВМ студента, если использовалось:*

Процессор **Intel(R) Core(TM) i7-12700H CPU @ 2.70GHz 2.50 GHz** с ОП **16109** Мб, ТТН **476 100** Мб. Встроенный монитор 13 дюймов диагональ, разрешение **2800 \* 1800**.

5. **Программное обеспечение:** *Программное обеспечение ПЭВМ студента, если использовалось****:***

Операционная система семейства **Ubuntu**, наименование версия **VirtualBox Ubuntu 20.04.3 LTS**,

интерпретатор команд **bash** версия **5.0.17**. Система программирования **C**.Редактор текстов **VI** версия **8.1**

6**. Идея, метод, алгоритм** *решения задачи [в формах: словесной, псевдокода, графической (блок-схема, диаграмма, рисунок, таблица) или формальные спецификации с пред- и постусловиями]:*

1. **Структура данных**:
   * Node: структура для узлов двусвязного списка, содержит данные (int), указатели на следующий (next) и предыдущий (prev) узлы.
   * List: структура для списка, содержит указатели на голову (head) и хвост (tail) списка.
2. **Операции со списком**:
   * createNode(int value): создает новый узел с заданным значением.
   * initList(List& list): инициализирует список, создавая фиктивные голову и хвост.
   * append(List& list, int value): добавляет новый узел с заданным значением в конец списка.
   * printList(const List& list): выводит элементы списка.
   * length(const List& list): возвращает количество элементов в списке.
3. **Задача task(List& list, int value, int k)**:
   * Проверяет текущую длину списка.
   * Если длина меньше k, добавляет в список k - currentLength элементов со значением value.
4. **Удаление узла**:
   * remove(List& list, Node\* node): удаляет указанный узел из списка, если это не фиктивные голова или хвост.
5. **Основная функция main()**:
   * Инициализирует список, добавляет несколько элементов.
   * Вызывает task() для изменения списка.
   * Выводит измененный список.
   * Удаляет первый элемент списка и снова выводит список.

7. **Сценарий выполнения работы** *(план работы, первоначальный текст программы в черновике (можно на отдельном листе) и тесты либо соображения по тестированию)*.

Сценарий выполнения работы:

1. Подготовить набор тестов.
2. Придумать алгоритм решения поставленной задачи.
3. Используя функции заданной системы программирования, реализовать придуманный алгоритм.
4. Провести тестирование программы, чтобы убедиться в корректности её работы.
5. Сформировать протокол.

Программа:

#include <iostream>

struct Node {

    int data;

    Node\* next;

    Node\* prev;

};

struct List {

    Node\* head;

    Node\* tail;

};

Node\* createNode(int value) {

    Node\* newNode = new Node;

    newNode->data = value;

    newNode->next = nullptr;

    newNode->prev = nullptr;

    return newNode;

}

void initList(List& list) {

    list.head = createNode(0);

    list.tail = createNode(0);

    list.head->next = list.tail;

    list.tail->prev = list.head;

}

void append(List& list, int value) {

    Node\* newNode = createNode(value);

    Node\* last = list.tail->prev;

    last->next = newNode;

    newNode->prev = last;

    newNode->next = list.tail;

    list.tail->prev = newNode;

}

void printList(const List& list) {

    Node\* current = list.head->next;

    while (current != list.tail) {

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

        current = current->next;

    }

    std::cout << std::endl;

}

int length(const List& list) {

    int count = 0;

    Node\* current = list.head->next;

    while (current != list.tail) {

        count++;

        current = current->next;

    }

    return count;

}

void task(List& list, int value, int k) {

    int currentLength = length(list);

    if (currentLength >= k) return;

    for (int i = currentLength; i < k; ++i) {

        append(list, value);

    }

}

void remove(List& list, Node\* node) {

    if (node == list.head || node == list.tail) return;

    node->prev->next = node->next;

    node->next->prev = node->prev;

    delete node;

}

int main() {

    List myList;

    initList(myList);

    append(myList, 10);

    append(myList, 20);

    append(myList, 30);

    task(myList, 0, 5);

    printList(myList);

    remove(myList, myList.head->next);

    printList(myList);

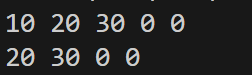
    return 0;

}

*Пункты 1-7 отчета составляются строго до начала лабораторной работы.*

*Допущен к выполнению работы.*  **Подпись преподавателя \_\_\_\_\_\_\_**

8. **Распечатка протокола** *(подклеить листинг окончательного варианта программы с тестовыми примерами, подписанный преподавателем).*



9. **Дневник отладки** должен содержать дату и время сеансов отладки и основные события (ошибки в сценарии и программе, нестандартные ситуации) и краткие комментарии к ним. В дневнике отладки приводятся сведения об использовании других ЭВМ, существенном участии преподавателя и других лиц в написании и отладке программы.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Лаб. или дом.** | **Дата** | **Время** | **Событие** | **Действие по исправлению** | **Примечание** |
|  |  |  |  |  |  |  |

10. **Замечания автора** по существу работы: замечания отсутствуют.

11. **Вывод:** в ходе выполнения данного задания практикума я научился работать с линейными списками в Си.

Подпись студента \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Отчёт по практикуму по циклу дисциплин «Алгоритмы и структуры данных».**

**Задание №9.**

Студент группы: **М8О-101Б-23**, **Терентьев Михаил Андреевич**, № по списку: 22, Контакты**,** [mihateren2003@gmail.com](mailto:mihateren2003@gmail.com), telegram **@mihteren** Работа выполнена: «09» мая 2024г.

Работа выполнена: «09» мая 2024 г.

Преподаватель: **каф. 805 Крылов Сергей Сергеевич**

Входной контроль знаний с оценкой: **\_\_\_\_\_**

Отчет сдан « » 2024 г.

Итоговая оценка: \_\_\_\_\_

Подпись преподавателя: \_\_\_\_\_\_\_\_\_\_

1. **Тема:** Сортировка и поиск

2. **Цель работы:** Составить программу на языке Си с использованием процедур и функций для сортировки

таблицы заданным методом и двоичного поиска по ключу в таблице.

3. **Задание:**

Составить программу на языке Си с использованием процедур и функций для сортировки таблицы заданным методом и двоичного поиска по ключу в таблице



4. **Оборудование**: *Оборудование ПЭВМ студента, если использовалось:*

Процессор **Intel(R) Core(TM) i7-12700H CPU @ 2.70GHz 2.50 GHz** с ОП **16109** Мб, ТТН **476 100** Мб. Встроенный монитор 13 дюймов диагональ, разрешение **2800 \* 1800**.

5. **Программное обеспечение:** *Программное обеспечение ПЭВМ студента, если использовалось****:***

Операционная система семейства **Ubuntu**, наименование версия **VirtualBox Ubuntu 20.04.3 LTS**,

интерпретатор команд **bash** версия **5.0.17**. Система программирования **C**.Редактор текстов **VI** версия **8.1**

6**. Идея, метод, алгоритм** *решения задачи [в формах: словесной, псевдокода, графической (блок-схема, диаграмма, рисунок, таблица) или формальные спецификации с пред- и постусловиями]:*

Реализовать линейный список для хранения таблицы, а также создать функции для сортировки таблицы.

1. Определить структуру данных для элемента таблицы.
2. Записать упорядоченную таблицу в файл вывода.
3. Запросить у пользователя ключ для поиска.
4. Выполнить двоичный поиск по ключу в упорядоченной таблице и вывести результат.
5. Закрыть файлы ввода и вывода.

7. **Сценарий выполнения работы** *(план работы, первоначальный текст программы в черновике (можно на отдельном листе) и тесты либо соображения по тестированию)*.

Программа:

#include <iostream>

using namespace std;

struct Record {

    int key;

    const char\* text;

};

int compareStrings(const char\* a, const char\* b) {

    while (\*a && (\*a == \*b)) {

        ++a;

        ++b;

    }

    return \*(const unsigned char\*)a - \*(const unsigned char\*)b;

}

void shellSort(Record table[], int n) {

    for (int gap = n / 2; gap > 0; gap /= 2) {

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

            Record temp = table[i];

            int j;

            for (j = i; j >= gap && compareStrings(table[j - gap].text, temp.text) > 0; j -= gap) {

                table[j] = table[j - gap];

            }

            table[j] = temp;

        }

    }

}

int binarySearch(Record table[], char key, int n) {

    int left = 0;

    int right = n - 1;

    while (left <= right) {

        int mid = left + (right - left) / 2;

        if (table[mid].text[0] == key) {

            return mid;

        }

        if (table[mid].text[0] < key) {

            left = mid + 1;

        } else {

            right = mid - 1;

        }

    }

    return -1;

}

void printTable(Record table[], int n) {

    for (int i = 0; i < n; i++) {

        cout << "Key: " << table[i].key << ", Text: " << table[i].text << endl;

    }

}

int main() {

    const char\* poem[] = {

        "Two roads diverged in a yellow wood,",

        "And sorry I could not travel both",

        "And be one traveler, long I stood",

        "And looked down one as far as I could",

        "To where it bent in the undergrowth;",

        "Then took the other, as just as fair,",

        "And having perhaps the better claim,",

        "Because it was grassy and wanted wear;",

        "Though as for that the passing there",

        "Had worn them really about the same,",

        "And both that morning equally lay"

    };

    const int n = sizeof(poem) / sizeof(poem[0]);

    Record table[n];

    for (int i = 0; i < n; i++) {

        table[i].key = i;

        table[i].text = poem[i];

    }

    shellSort(table, n);

    cout << "Sorted table by the first letter of each line:" << endl;

    printTable(table, n);

    char keyToFind = 'T'; // Пример ключа для поиска (первая буква первой строки)

    int index = binarySearch(table, keyToFind, n);

    if (index != -1) {

        cout << "Record found: Key: " << keyToFind << ", Text: " << table[index].text << endl;

    } else {

        cout << "Record with key " << keyToFind << " not found." << endl;

    }

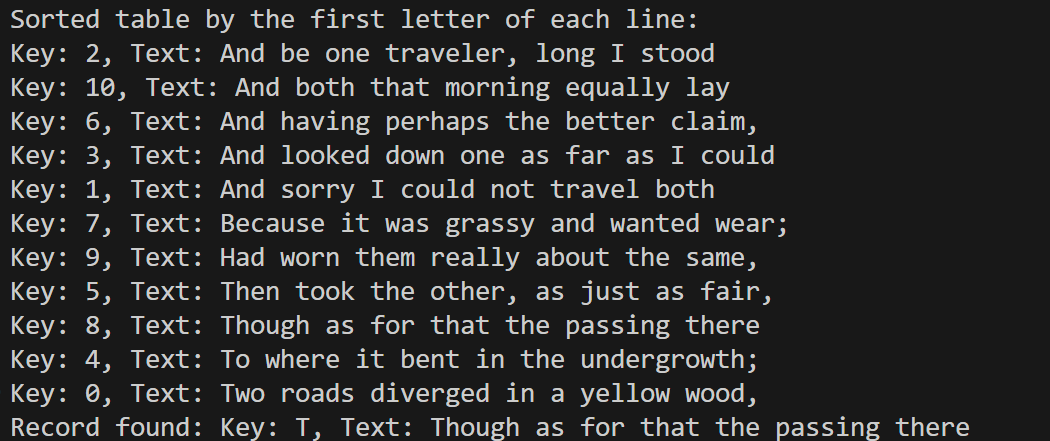
    return 0;

}

*Пункты 1-7 отчета составляются строго до начала лабораторной работы.*

*Допущен к выполнению работы.*  **Подпись преподавателя \_\_\_\_\_\_\_**

8. **Распечатка протокола** *(подклеить листинг окончательного варианта программы с тестовыми примерами, подписанный преподавателем).*



9. **Дневник отладки** должен содержать дату и время сеансов отладки и основные события (ошибки в сценарии и программе, нестандартные ситуации) и краткие комментарии к ним. В дневнике отладки приводятся сведения об использовании других ЭВМ, существенном участии преподавателя и других лиц в написании и отладке программы.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Лаб. или дом.** | **Дата** | **Время** | **Событие** | **Действие по исправлению** | **Примечание** |
|  |  |  |  |  |  |  |

10. **Замечания автора** по существу работы: замечания отсутствуют.

11. **Вывод:** в ходе выполнения данного задания практикума я научилась работать с линейными списками в СП Си.

Подпись студента \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_