|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ | | | | | | | | | | | | |
| Федеральное государственное автономное образовательное учреждение  высшего образования | | | | | | | | | | | | |
| **Дальневосточный федеральный университет** | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| **ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ** | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| **Департамент программной инженерии и искусственного интеллекта** | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| **О Т Ч Е Т** | | | | | | | | | | | | |
| по лабораторной работе №1.1  дисциплина «Фундаментальные структуры данных и алгоритмы» | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | |  |  | Студент гр. Б9122-09.03.04прогин | | | |
|  |  |  | | А.О. Лопатин | |
|  | | | | | | |  |  | (подпись) | |  | |
|  | | | | | | | | | | | | |
|  | | | | | | |  |  | Руководитель | | |  |
|  | | | | | | |  |  | ст. преподаватель | | | |
|  | | | |  |  | |  |  |  |  | О.А. Крестникова | |
|  | | | |  |  | |  |  | (подпись) |  | (И.О. Фамилия) | |
|  |  |  |  | | |  |  |  |  | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| г. Владивосток | | | | | | | | | | | | |
| 2023 | | | | | | | | | | | | |

1. **Неформальная постановка задачи**

Реализовать пакет подпрограмм для работы с бинарным деревом поиска(несбалансированным), ключ в котором представлен в виде структуры (ФИО – 3 поля строкового типа)

1. **Описание типов и спецификация подпрограмм с тестами**

**Тип** Key

- Ключ – 3 поля строкового типа: имя, фамилия, отчество.

**Тип** numlist

1 поле - указатель на подтип Node типа numlist.

### **Тип** Node

узел с unsigned short int полем и указателем на node

### **Функции**

#### Node\* searchNum(int \_num)

Возвращает адрес узла со значением \_num

Входные данные: int \_num

Выходные данные: Node\*

|  |  |  |
| --- | --- | --- |
| Описание тестовой ситуации | Входные данные | Выходные данные |
| Ищем существующее в списке значение | int \_num | Node\* |
| Ищем несуществующее в списке значение | int \_num | nullptr |

### **Процедуры**

#### void pushBack(int \_num)

добавляет новый узел, с заданным значением, в конец списка

Входные данные: int \_num

|  |  |  |
| --- | --- | --- |
| Описание тестовой ситуации | Входные данные | Выходные данные |
| Вставляем значение в пустой список | int \_num | numlist |
| Вставляем значение в непустой список | int \_num | numlist |

#### void print\_f(std::basic\_ofstream<char> &file)

печатает список в файл

Входные данные: file

|  |  |  |
| --- | --- | --- |
| Описание тестовой ситуации | Входные данные | Выходные данные |
| Печатаем пустой список | numlist, file | - |
| Печатаем непустой список | numlist , file | 1,2,3… - элементы numlist в file |

#### void deleteList()

добавляет новый узел, с заданным значением, в конец списка

Входные данные: int \_num

|  |  |  |
| --- | --- | --- |
| Описание тестовой ситуации | Входные данные | Выходные данные |
| Удаляем пустой список | numlist | - |
| Удаляем непустой список | numlist | - |

## **Тип** Tree

Дерево – 1 поле – указатель на подтип Node типа tree

### Тип Node

узел с полем Ключа и указателями на родителя, левого и правого потомков (Node\*), списком номеров.

### **Функции:**

#### Node\* search\_max\_in\_left\_subtree (Node\* subtree)

Возвращает максимальный элемент в левом поддереве для входящей вершины

Входные данные: Указатель на вершину поддерева

Выходные данные: Указатель на вершину поддерева

|  |  |  |
| --- | --- | --- |
| Описание тестовой ситуации | Входные данные | Выходные данные |
| На вход подается вершина без левых потомков | Node\* subtree | null |
| На вход подается вершина c левым потомком | Node\* subtree | Node\* leftM = subtree->left |

#### Node\* searchElem (Key \_fio)

Возвращает адрес на элемент с ключом \_fio

Входные данные: ключ

Выходные данные: Указатель на вершину поддерева

|  |  |  |
| --- | --- | --- |
| Описание тестовой ситуации | Входные данные | Выходные данные |
| Подается ключ, который является ключом одного из узлов дерева | Key \_fio | Node\* subtree |
| Подается ключ, которого нет в дереве | Key \_fio | nullptr |

#### char compare(Node\* subtree, Key \_fio)

возвращает ‘<’, ‘>’, ‘=’ в зависимости от алфавитного сравнения subtree->fio и \_fio.

Входные данные: ключ

Выходные данные: символ

|  |  |  |
| --- | --- | --- |
| Описание тестовой ситуации | Входные данные | Выходные данные |
| Подается ключ, который равный ключу вершины | Key \_fio  Node\* subtree | ‘=’ |
| Подается ключ, алфавитно больший ключа вершины | Key \_fio  Node\* subtree | ‘<’ |
| Подается ключ, алфавитно меньший  ключа вершины | Key \_fio  Node\* subtree | ‘>’ |

#### bool check\_fio(std::string anyname)

возвращает True, если строка валидна(только латиница), иначе False

Входные данные: строка

Выходные данные: char

|  |  |  |
| --- | --- | --- |
| Описание тестовой ситуации | Входные данные | Выходные данные |
| Подается cтрока содержащая только латиницу | anyname | True |
| Подается строка, содержащая символ отличный от латиницы | anyname | false |

### **Процедуры:**

#### void addElem(Node\* subtree, Key \_fio, unsigned short int num)

рекурсивно добавляет элемент в дерево по ключу, записывает его номер в

supersubtree->repeats.

Входные данные: вершина дерева

|  |  |  |
| --- | --- | --- |
| Описание тестовой ситуации | Входные данные | Выходные данные |
| Добавление узла с ключом, совпадающим с каким-либо в дереве | Tree.root, key \_fio, number | Subtree->repeats.push\_back(number) |
| Добавление узла с уникальным ключом | Tree.root, key \_fio, number | Subtree->fio = \_fio;  Subtree->repeats.push\_back(number) |

#### void deleteElem(Key \_fio)

удаляет элемент в дереве по ключу

Входные данные: ключ

|  |  |  |
| --- | --- | --- |
| Описание тестовой ситуации | Входные данные | Выходные данные |
| Удаление ФИО, не существующего в дереве | key \_fio | - |
| Удаление ФИО, существующего в дереве | key \_fio | - |

#### void deleteTree(Node\* subtree)

удаляет дерево

Входные данные: вершина дерева

|  |  |  |
| --- | --- | --- |
| Описание тестовой ситуации | Входные данные | Выходные данные |
| Удаление пустого дерева | Tree.root | - |
| Удаление непустого дерева | Tree.root, key \_fio, number | - |

#### void goThrough\_f(Node\* subtree, std::basic\_ofstream<char>& file)

печатает элементы дерева в порядке обхода «слева направо»

Входные данные: вершина дерева, файл

|  |  |  |
| --- | --- | --- |
| Описание тестовой ситуации | Входные данные | Выходные данные |
| Печать пустого дерева | Tree.root, file | - |
| Печать непустого дерева | Tree.root, key \_fio, number | a, b, …, где a, b – ключи элементов дерева. |

#### void print\_f(Node\* subtree, std::basic\_ofstream<char>& file, int space = 0)

печатает дерево, повернутое на 90 градусов против часовой стрелки, в файл.

Входные данные: вершина дерева, файл.

|  |  |  |
| --- | --- | --- |
| Описание тестовой ситуации | Входные данные | Выходные данные |
| Печать пустого дерева | Tree.root, file | - |
| Печать непустого дерева | Tree.root, key \_fio, number | Граф дерева |

#### void read\_f(std::ifstream &fin)

считывает элементы дерева с файла.

Входные данные: вершина дерева

|  |  |  |
| --- | --- | --- |
| Описание тестовой ситуации | Входные данные | Выходные данные |
| Считывание пустого файла | fin | - |
| Считывание неправильного файла | fin | - |
| Считывание валидного файла | fin | tree |

1. **Текст программы**