|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 №1.2  дисциплина «Фундаментальные структуры данных и алгоритмы» | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | |  |  | Студент гр. Б9122-09.03.04прогин | | | |
|  |  |  | | А.В. Поляков | |
|  | | | | | | |  |  | (подпись) | |  | |
|  | | | | | | | | | | | | |
|  | | | | | | |  |  | Руководитель | | |  |
|  | | | | | | |  |  | ст. преподаватель | | | |
|  | | | |  |  | |  |  |  |  | О.А. Крестникова | |
|  | | | |  |  | |  |  | (подпись) |  | (И.О. Фамилия) | |
|  |  |  |  | | |  |  |  |  | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| г. Владивосток | | | | | | | | | | | | |
| 2023 | | | | | | | | | | | | |

**1 Неформальная постановка задачи**

Реализовать функции для работы с двусвязным кольцевым списком.

Реализовать функции для работы с бинарным деревом. Ключ дерева – код направления.

Функции:

1. Добавление: добавление в дерево происходит по принципу «слева – меньше, справа – больше», например, в дерево с корнем Б9000 левым потомком Б8000 и правым потомком Б9500 добавляем Б9300 и у листа Б9500 появляется левый потомок Б9300.
2. Удаление: при удалении идёт замена на минимальный справа. Например, из дерева с корнем Б9000 левым потомком Б8000, правым потомком Б9500, у которого есть лист Б9300 удаляем корень Б9000. Происходит замена на минимальный справа: Б9300 становится новым корнем.
3. Распечатка дерева на экран: выводит дерево на экран.
4. Поиск заданного элемента в дереве: выводит ключ и номера строк, в которых находится заданный ключ
5. Удаление всего дерева: удаление всего дерева.
6. Обход дерева: симметричный, справа налево.

Входные данные: текстовый файл DataIn.txt

Выходные данные: текстовый файл DataOut.txt

**2 Описание типа + спецификация подпрограмм + тесты**

struct StudentGroup {

char grade;

int smthAndYear;

StudentGroup(string name) : grade(name[0]), smthAndYear(stoi(name.substr(1, 4))) {}

bool operator < (const StudentGroup& other){

return grade < other.grade || (grade == other.grade && smthAndYear < other.smthAndYear);

}

bool operator > (const StudentGroup& other){

return grade > other.grade || (grade == other.grade && smthAndYear > other.smthAndYear);

}

bool operator == (const StudentGroup& other){

return grade == other.grade && smthAndYear == other.smthAndYear;

}

};

struct NodeList {

int data;

NodeList\* prev;

NodeList\* next;

};

struct NodeTree {

StudentGroup key;

NodeTree\* left;

NodeTree\* right;

NodeList\* list;

NodeTree(StudentGroup value) : key(value), left(nullptr), right(nullptr), list(nullptr) {}

};

Структура StudentGroup. Имеет 2 поля char grade – буква, которая обозначает программу обучения и int smthAndYear – 4 цифры после буквы со своим смыслом. Содержит в себе базовый конструктор и перегруженные операции сравнения.

Структура NodeList имеет поля: data типа int – хранит целое число в информационном поле, указатель на next типа NodeList – хранит указатель на следующий узел списка, указатель на prev типа NodeList – хранит указатель на предыдущий узел списка.

Структура дерева. Имеет поля: key типа StudentGroup, которое хранит значение кода направления, указатели на left и right типа NodeTree, которые указывают на потомков и указатель на поле list типа NodeList, который хранит номера строк ключей во входном файле.

**function** insert(NodeTree\*& root, StudentGroup value, int lineNumber): NodeTree\*

Функция для вставки узла дерева с колючем value и номером строки lineNumber в бинарное дерево. Также вставляется в список номер строки с помощью функции appendToBegining(NodeList\*& head, int value).

Выходные данные: дерево после процедуры вставки элемента, 1 – успешное добавление.

|  |  |  |
| --- | --- | --- |
| **Описание тестовой ситуации** | **Входные данные** | **Выходные данные** |
| Добавление в пустое дерево | root (root == nullptr), Б5000 | Б5000 1 |
| Добавление в непустое дерево | Root (  Б5000 1  Б3000 2 Б7000 3) , Б9000 | Б5000 1  Б3000 2 Б7000 3  Б9000 4 |
| Добавление элемента, который уже есть в дереве (повторный) | Root (  Б5000 1  Б3000 2 Б7000 3  Б9000 4  ) , Б9000 | Б5000 1  Б3000 2 Б7000 3  Б9000 4 5 |

**function** delMinRight(NodeTree\*& root, NodeTree\*& q): void

Функция для рекурсивного нахождения и последующего удаления минимального элемента в правом поддереве.

**function** remove(NodeTree\*& root, StudentGroup value, int lineNumber): void

Функция для удаления узла из дерева. Если узел пуст, тогда ничего не удаляется. Если у узла ребенок, тогда удаляется выбранный узел, а потомок подцепляется к деду. Если у узла 2 ребенка, тогда с помощью функции delMinRight находим минимальный справа, удаляем выбранный узел, меняем его на найденный минимальный справа.

Входные данные: корень дерева, ключ, который надо удалить, номер строки, в котором он находится.

|  |  |  |
| --- | --- | --- |
| **Описание тестовой ситуации** | **Входные данные** | **Выходные данные** |
| Удаление из пустого дерева | root (root == nullptr) |  |
| Удаление листа | Root (  Б5000 1  Б3000 2 Б7000 3) , Б7000 3 | Б5000 1  Б3000 2 |
| Удаление узла с 1 потомком | Root (  Б5000 1  Б3000 2 Б7000 3  Б9000 4), Б7000 3 | Б5000 1  Б3000 2 Б9000 4 |
| Удаление узла с двумя детьми | Root (  Б5000 1  Б3000 2 Б7000 3  Б9000 4 5)  , Б5000 1 | Б7000 3  Б3000 2 Б9000 4 5 |
| Удаление из непустого дерева, элемента, который несколько раз встречался во входном файле (удаляется выбранный номер строки) | Root (  Б5000 1  Б3000 2 Б7000 3  Б9000 4 5)  , Б9000 4 | Б5000 1  Б3000 2 Б7000 3  Б9000 5 |
| Удаление элемента, которого нет в дереве | Root (  Б5000 1  Б3000 2 Б7000 3) , Б9000 4 | Б5000 1  Б3000 2 Б7000 3 |

**function** symmetricTraversalToFile(NodeTree\* root, ofstream& outFile): void

Функция для занесения ключей дерева и номеров строк в выходной файл в порядке симметричного обхода справа налево.

Входные данные: 1) указатель на корень дерева, 2) ссылка на выходной файл. Выходные данные: занесение дерева (его ключей) в порядке симметричного обхода справа налево и номеров строк в выходной файл.

|  |  |  |
| --- | --- | --- |
| **Описание тестовой ситуации** | **Входные данные** | **Выходные данные** |
| Непустое дерево | Root (  Б5000 1  Б3000 2 Б7000 3 | Б7000 3  Б5000 1  Б3000 2 |
| Пустое дерево | Root (root == nullptr) |  |

**function** searchKey(NodeTree\* root, StudentGroup value): NodeList\*

Рекурсивная функция для поиска ключа value в бинарном дереве root и вывод связанного списка, если ключ найден. Если ключ найден, выводит сообщение «Ключ найден.» и печатает весь список, принадлежащий этому ключу. Если ключ не найден в дереве, тогда выводит сообщение «Ключ не найден.»

**function** draw(NodeTree\* root, int h): void

Функция для печати повернутого на 90 градусов дерева.

Пример работы:  
 B9007 (19 12 5 )

B9006 (21 14 7 )

B9005 (20 13 6 )

B9004 (15 8 1 )

B9003 (18 11 4 )

B9002 (16 9 2 )

B9001 (17 10 3 )

B9004 – корень. B9002 и B9007 – дети B9004 и так далее.

**function** deleteListForKey(NodeTree\* root, StudentGroup key): void

Функция для отчистки списка, принадлежащего выбранному ключу.

Если не удается найти ключ, выводится сообщение: «Ключ не найден в дереве.» Если ключ найден, то удаляется его список.

|  |  |  |
| --- | --- | --- |
| **Описание тестовой ситуации** | **Входные данные** | **Выходные данные** |
| Ключ не найден | Root (  Б5000 1  Б3000 2 Б7000 3), Б9000 | Ключ не найден в дереве. |
| Ключ найден | Root (  Б5000 1  Б3000 2 Б7000 3 4), Б7000 | Б7000 () |

**3 Текст программы**

#include <iostream>

#include <iomanip>

#include <fstream>

#include <string>

using namespace std;

struct StudentGroup {

char grade;

int smthAndYear;

StudentGroup(string name) : grade(name[0]), smthAndYear(stoi(name.substr(1, 4))) {}

bool operator < (const StudentGroup& other){

return grade < other.grade || (grade == other.grade && smthAndYear < other.smthAndYear);

}

bool operator > (const StudentGroup& other){

return grade > other.grade || (grade == other.grade && smthAndYear > other.smthAndYear);

}

bool operator == (const StudentGroup& other){

return grade == other.grade && smthAndYear == other.smthAndYear;

}

};

struct NodeList {

int data;

NodeList\* prev;

NodeList\* next;

};

struct NodeTree {

StudentGroup key;

NodeTree\* left;

NodeTree\* right;

NodeList\* list;

NodeTree(StudentGroup value) : key(value), left(nullptr), right(nullptr), list(nullptr) {}

};

NodeList\* createNode(int value) {

NodeList\* newNode = new NodeList;

newNode->data = value;

newNode->prev = nullptr;

newNode->next = nullptr;

return newNode;

}

bool findNodeList(NodeList\* head, int value){

NodeList\* current = head;

do{

if (current->data == value) {

return true;

}

current = current->next;

}while(current != head);

return false;

}

void deleteChosenElement(NodeList\*& head, int value) {

NodeList\* current = head;

if(current == nullptr)

return;

do{

if(current->next == current && current->data == value){

delete head;

head = nullptr;

current = nullptr;

return;

}

else if(current->data == value){

bool isHead = false;

if(current == head)

isHead = true;

NodeList\* delEl = current;

current = current->next;

delEl->prev->next = current;

current->prev = delEl->prev;

if (isHead) {

head = current;

}

delete delEl;

delEl = nullptr;

}

else{

current = current->next;

}

}while(current != head);

}

void appendToBegining(NodeList\*& head, int value) {

NodeList\* newNode = createNode(value);

if (!head) {

head = newNode;

head->next = head;

head->prev = head;

} else {

NodeList\* tail = head->prev;

newNode->next = head;

newNode->prev = tail;

head->prev = newNode;

tail->next = newNode;

head = newNode;

}

}

NodeTree\* insert(NodeTree\*& root, StudentGroup value, int lineNumber) {

if (root == nullptr) {

NodeTree\* newNode = new NodeTree(value);

appendToBegining(newNode->list, lineNumber);

return newNode;

}

if ((value.grade < root->key.grade) ||

(value.grade == root->key.grade && value.smthAndYear < root->key.smthAndYear)) {

root->left = insert(root->left, value, lineNumber);

} else if ((value.grade > root->key.grade) ||

(value.grade == root->key.grade && value.smthAndYear > root->key.smthAndYear)) {

root->right = insert(root->right, value, lineNumber);

} else {

appendToBegining(root->list, lineNumber);

}

return root;

}

// Для удаления

void deleteList(NodeList\*& head) {

if (head != nullptr) {

NodeList\* current = head;

NodeList\* next;

do {

NodeList\* temp = current;

current = current->next;

delete temp;

} while (current != head);

head = nullptr;

}

}

void clearNode(NodeList\*& head) {

NodeList\* current = head;

if (!head) {

std::cout << "Список пуст. Чистить нечего." << endl;

return;

}

do {

NodeList\* temp = current;

current = current->next;

delete temp;

} while (current != head);

}

void delMinRight(NodeTree\*& root, NodeTree\*& q){

if(root->left != nullptr){

delMinRight(root->left, q);

} else{

q->key = root->key;

q->list = root->list;

root = root->right;

}

}

void remove(NodeTree\*& root, StudentGroup value, int lineNumber) {

if (root == nullptr) {

return;

}

if (value < root->key) {

remove(root->left, value, lineNumber);

} else if (value > root->key) {

remove(root->right, value, lineNumber);

}

if(findNodeList(root->list, lineNumber)){

// Если только один элемент в списке

if(root->list->next != root->list){

deleteChosenElement(root->list, lineNumber);

}

else{

// Если более одного, удаляет номер строки

if(root->left == nullptr){

root = root->right;

}

else if(root->right == nullptr){

root = root->left;

}

else{

NodeTree\* q = root;

delMinRight(q->right, q);

}

}

}

}

void printNodeList(NodeList\* head) {

if (head) {

NodeList\* current = head;

do {

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

current = current->next;

} while (current != head);

} else {

cout << "Список пуст.";

return;

}

}

void printAllNodeLists(NodeTree\* root) {

if (root != nullptr) {

cout << root->key.grade << setw(4) << setfill('0') << root->key.smthAndYear << ": ";

printNodeList(root->list);

cout << endl;

printAllNodeLists(root->left);

printAllNodeLists(root->right);

}

}

void symmetricTraversalToFile(NodeTree\* root, ofstream& outFile) {

if (root != nullptr) {

symmetricTraversalToFile(root->left, outFile);

NodeList\* current = root->list;

do {

outFile << root->key.grade << setw(4) << setfill('0') << root->key.smthAndYear << endl;

current = current->next;

} while (current != root->list);

outFile << setw(0) << setfill(' ');

symmetricTraversalToFile(root->right, outFile);

}

}

NodeList\* searchKey(NodeTree\* root, StudentGroup value) {

if (root == nullptr || (root->key.grade == value.grade && root->key.smthAndYear == value.smthAndYear)) {

if (root != nullptr) {

if (root->list != nullptr) {

cout << "Ключ найден. " << root->key.grade << setw(4) << setfill('0') << root->key.smthAndYear << ": ";

printNodeList(root->list);

cout << endl;

} else {

cout << "Ключ найден. " << root->key.grade << setw(4) << setfill('0') << root->key.smthAndYear << ": Список пуст" << endl;

}

return root->list;

} else {

cout << "Ключ не найден." << endl;

return nullptr;

}

}

if (value.grade < root->key.grade || (value.grade == root->key.grade && value.smthAndYear < root->key.smthAndYear)) {

return searchKey(root->left, value);

} else {

return searchKey(root->right, value);

}

}

void draw(NodeTree\* root, int h) {

if (root != nullptr) {

draw(root->right, h + 4);

for (int i = 1; i <= h; i++) {

cout << " ";

}

cout << root->key.grade << setw(4) << setfill('0') << root->key.smthAndYear;

cout << " (";

printNodeList(root->list);

cout << ")" << endl;

draw(root->left, h + 4);

}

}

void deleteListForKey(NodeTree\* root, StudentGroup key) {

NodeTree\* node = root;

while (node != nullptr) {

if (key.grade < node->key.grade || (key.grade == node->key.grade && key.smthAndYear < node->key.smthAndYear)) {

node = node->left;

} else if (key.grade > node->key.grade || (key.grade == node->key.grade && key.smthAndYear > node->key.smthAndYear)) {

node = node->right;

} else {

NodeList\* current = node->list;

NodeList\* next;

do {

NodeList\* temp = current;

current = current->next;

delete temp;

} while (current != node->list);

node->list = nullptr;

return;

}

}

cout << "Ключ не найден в дереве." << endl;

}

void deleteTree(NodeTree\*& root) {

if (root != nullptr) {

deleteTree(root->left);

deleteTree(root->right);

if (root->list) {

deleteList(root->list);

}

delete root;

root = nullptr;

}

}

int main() {

NodeTree\* root = nullptr;

ifstream inFile("C:/Users/User/Desktop/FEFU/Second\_course/FDSA/DataIn.txt");

string input;

int lineNumber = 1;

while (inFile >> input) {

root = insert(root, StudentGroup(input), lineNumber);

lineNumber++;

}

inFile.close();

ofstream outFile("C:/Users/User/Desktop/FEFU/Second\_course/FDSA/DataOut.txt");

symmetricTraversalToFile(root, outFile);

outFile << endl;

outFile.close();

draw(root, 0);

cout << endl;

StudentGroup keyForDelete("B9007");

remove(root, keyForDelete, 5);

remove(root, keyForDelete, 12);

remove(root, keyForDelete, 19);

draw(root, 0);

return 0;

}