**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра АПУ**

**ОТЧЕТ**

**по лабораторной работе №5**

**на тему: «АВЛ-дерево»**

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

|  |  |  |
| --- | --- | --- |
| Студент гр. 3391 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Николаев В.Ю. |
| Преподаватель | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Ряскова Е.Б. |

Санкт-Петербург

2024 г.

## Цель работы

Получить практические навыки реализации и применения алгоритмов для построения и работы с АВЛ-деревом.

## Задачи

Создать программу, которая:

* Реализует АВЛ-дерево для хранения информации о студентах.
* Включает следующие функции: добавление, вывод, поиск, удаление информации о студенте.
* Обеспечивает диалог с пользователем через меню для выполнения операций с деревом.

## Описание программы

Программа предоставляет пользователю возможность хранить и обрабатывать данные студентов с помощью АВЛ-дерева. Реализованы следующие функции:

* Добавление студента: ввод данных студента (№ студенческого билета, ФИО, курс, дата зачисления, контактная информация) и добавление их в АВЛ-дерево.
* Вывод информации: визуализация АВЛ-дерева (можно использовать любой удобный метод отображения дерева).
* Поиск студента по номеру студенческого билета: вывод всех данных студента, если он найден.
* Удаление информации: удаление записи о студенте по номеру студенческого билета.
* Очистка дерева: удаление всех записей.

Листинг кода программы приведён в Приложении 1.

## Алгоритмы

### Ввод данных

Пользователь вводит данные о студенте через интерфейс программы. Для каждого студента необходимо ввести следующие данные:

* Номер студенческого билета (ключ для АВЛ-дерева).
* ФИО студента.
* Курс.
* Дата зачисления.
* Контактная информация.

В программе будет реализована функция для ввода этих данных, которые затем добавляются в дерево с использованием функции добавления.

### Добавление в АВЛ-дерево

АВЛ-дерево — это самобалансирующееся бинарное дерево поиска. Основная его особенность заключается в поддержании сбалансированности дерева при добавлении и удалении элементов, чтобы время поиска, добавления и удаления оставалось логарифмическим — O(log n).

Алгоритм добавления в АВЛ-дерево включает следующие шаги:

1. **Вставка узла**: Как и в обычном бинарном дереве поиска, новый узел добавляется в то место, где нарушается свойство порядка. Для этого поочередно сравнивается значение номера студенческого билета (ключ) нового студента с ключами узлов, начиная с корня дерева. Если новый ключ меньше, переход осуществляется к левому поддереву; если больше — к правому.
2. **Проверка баланса**: После вставки узла проверяется балансировка всех узлов, начиная с узла, в который было вставлено новое значение. Важный параметр — разница высот левого и правого поддеревьев каждого узла (баланс-фактор). Если она больше 1 или меньше -1, балансировка нарушена.
3. **Балансировка**: Если балансировка нарушена, выполняется одно из четырех вращений:
   * **Одно правое вращение** (если вставка произошла в левое поддерево левого поддерева).
   * **Одно левое вращение** (если вставка произошла в правое поддерево правого поддерева).
   * **Левое-правое вращение** (если вставка произошла в правое поддерево левого поддерева).
   * **Правое-левое вращение** (если вставка произошла в левое поддерево правого поддерева).

Эти вращения восстанавливают баланс дерева и сохраняют его как сбалансированное бинарное дерево поиска.

### Поиск студента

Поиск в АВЛ-дереве выполняется так же, как и в обычном бинарном дереве поиска:

1. Поиск начинается с корневого узла дерева. Номер студенческого билета нового студента сравнивается с ключом корневого узла.
2. Если номер совпадает — найден нужный узел, и программа выводит все данные о студенте.
3. Если ключ меньше, переход осуществляется в левое поддерево, если больше — в правое.
4. Процесс повторяется до тех пор, пока не будет найден нужный узел, или пока не достигнется лист дерева, что означает, что студента с данным номером в дереве нет.

Поиск в АВЛ-дереве занимает время O(log n), поскольку дерево сбалансировано.

### Удаление записи

Удаление узла из АВЛ-дерева выполняется следующим образом:

1. **Поиск удаляемого узла**: Сначала находится узел, который нужно удалить, по ключу (номеру студенческого билета), аналогично процессу поиска.
2. **Удаление узла**: Есть три возможных случая:
   * **Узел — лист**: Просто удаляется узел.
   * **У узла одно поддерево**: Узел удаляется, и его поддерево заменяет его.
   * **У узла два поддерева**: Необходимо найти узел с минимальным значением в правом поддереве (преемник), заменить значение удаляемого узла на значение преемника и удалить преемник.
3. **Проверка баланса**: После удаления, как и в случае добавления, необходимо проверить балансировку дерева на всех уровнях, начиная с удаленного узла. Если баланс нарушен, выполняются соответствующие вращения для восстановления балансировки.

### Очистка дерева

Функция очистки удаляет все узлы дерева, тем самым освобождая память, выделенную под них. Алгоритм может выполняться как постфиксный обход дерева:

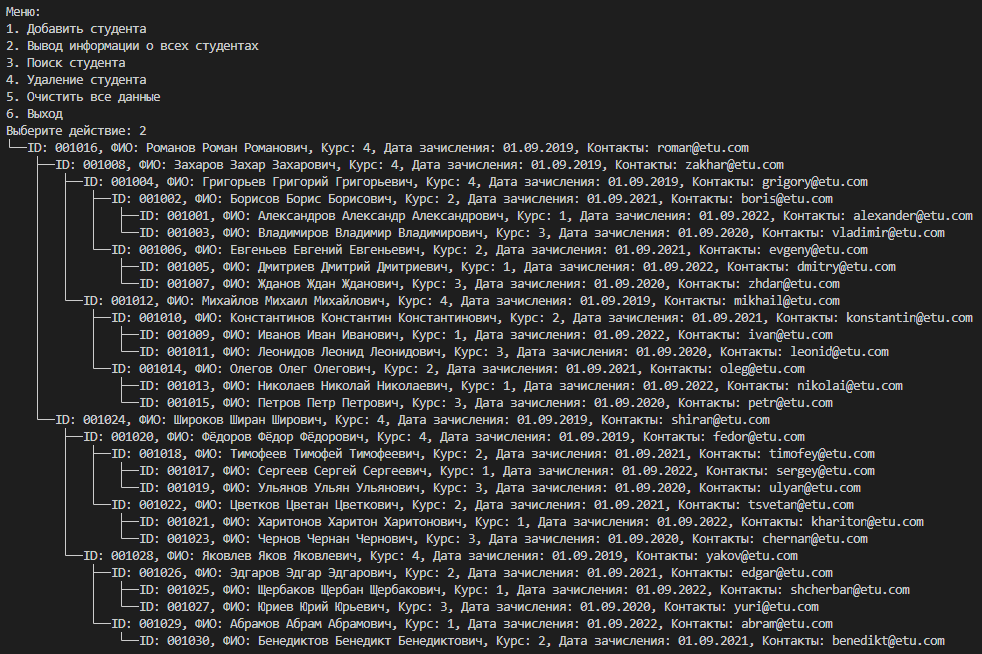
* Для каждого узла вызывается рекурсивное удаление его поддеревьев.
* Затем удаляется сам узел.

Очистка дерева освобождает всю память и обнуляет указатель на корень дерева, тем самым делая дерево пустым.

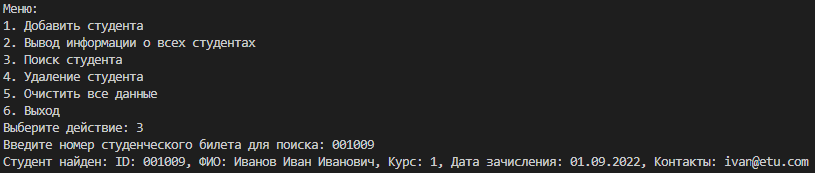
## Результаты работы программы

Для проверки работы программы используем следующий массив студентов:

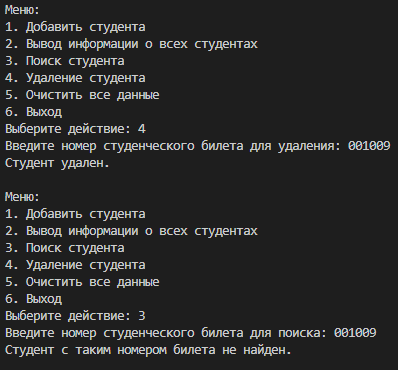
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **ФИО** | **Курс** | **Дата зачисления** | **Контакты** |
| 001001 | Александров Александр Александрович | 1 | 01.09.2022 | alexander@etu.com |
| 001002 | Борисов Борис Борисович | 2 | 01.09.2021 | boris@etu.com |
| 001003 | Владимиров Владимир Владимирович | 3 | 01.09.2020 | vladimir@etu.com |
| 001004 | Григорьев Григорий Григорьевич | 4 | 01.09.2019 | grigory@etu.com |
| 001005 | Дмитриев Дмитрий Дмитриевич | 1 | 01.09.2022 | dmitry@etu.com |
| 001006 | Евгеньев Евгений Евгеньевич | 2 | 01.09.2021 | evgeny@etu.com |
| 001007 | Жданов Ждан Жданович | 3 | 01.09.2020 | zhdan@etu.com |
| 001008 | Захаров Захар Захарович | 4 | 01.09.2019 | zakhar@etu.com |
| 001009 | Иванов Иван Иванович | 1 | 01.09.2022 | ivan@etu.com |
| 001010 | Константинов Константин Константинович | 2 | 01.09.2021 | konstantin@etu.com |
| 001011 | Леонидов Леонид Леонидович | 3 | 01.09.2020 | leonid@etu.com |
| 001012 | Михайлов Михаил Михайлович | 4 | 01.09.2019 | mikhail@etu.com |
| 001013 | Николаев Николай Николаевич | 1 | 01.09.2022 | nikolai@etu.com |
| 001014 | Олегов Олег Олегович | 2 | 01.09.2021 | oleg@etu.com |
| 001015 | Петров Петр Петрович | 3 | 01.09.2020 | petr@etu.com |
| 001016 | Романов Роман Романович | 4 | 01.09.2019 | roman@etu.com |
| 001017 | Сергеев Сергей Сергеевич | 1 | 01.09.2022 | sergey@etu.com |
| 001018 | Тимофеев Тимофей Тимофеевич | 2 | 01.09.2021 | timofey@etu.com |
| 001019 | Ульянов Ульян Ульянович | 3 | 01.09.2020 | ulyan@etu.com |
| 001020 | Фёдоров Фёдор Фёдорович | 4 | 01.09.2019 | fedor@etu.com |
| 001021 | Харитонов Харитон Харитонович | 1 | 01.09.2022 | khariton@etu.com |
| 001022 | Цветков Цветан Цветкович | 2 | 01.09.2021 | tsvetan@etu.com |
| 001023 | Чернов Чернан Чернович | 3 | 01.09.2020 | chernan@etu.com |
| 001024 | Широков Ширан Ширович | 4 | 01.09.2019 | shiran@etu.com |
| 001025 | Щербаков Щербан Щербакович | 1 | 01.09.2022 | shcherban@etu.com |
| 001026 | Эдгаров Эдгар Эдгарович | 2 | 01.09.2021 | edgar@etu.com |
| 001027 | Юриев Юрий Юрьевич | 3 | 01.09.2020 | yuri@etu.com |
| 001028 | Яковлев Яков Яковлевич | 4 | 01.09.2019 | yakov@etu.com |
| 001029 | Абрамов Абрам Абрамович | 1 | 01.09.2022 | abram@etu.com |
| 001030 | Бенедиктов Бенедикт Бенедиктович | 2 | 01.09.2021 | benedikt@etu.com |



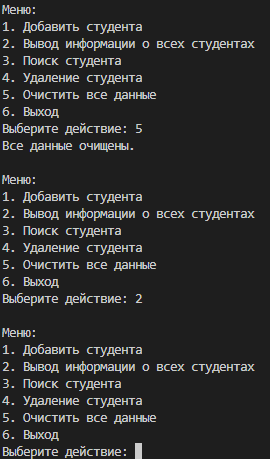
Найдём студента Иванова Ивана Ивановича (ID: 001009):



Отчислим этого студента:



Отчистим данные:



## Выводы

В ходе выполнения лабораторной работы были получены навыки построения и работы с АВЛ-деревом. Были реализованы различные операции с деревом, такие как добавление, поиск, удаление и балансировка.

## Приложение 1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| AVLTree.h  |  | | --- | | #ifndef AVL\_TREE\_H  #define AVL\_TREE\_H  #include "Student.h"  struct Node {  Student data;  Node\* left;  Node\* right;  int height;  Node(const Student& student);  };  class AVLTree {  private:  Node\* root;  Node\* insert(Node\* node, const Student& student);  Node\* deleteNode(Node\* node, const char\* student\_id);  Node\* find(Node\* node, const char\* student\_id);  void clear(Node\* node);  void printTreeHelper(Node\* node, const std::string& prefix, bool isLeft);  int getHeight(Node\* node);  int getBalanceFactor(Node\* node);  Node\* rightRotate(Node\* node);  Node\* leftRotate(Node\* node);  Node\* minValueNode(Node\* node);  public:  AVLTree();  ~AVLTree();  void insert(const Student& student);  void deleteStudent(const char\* student\_id);  Student\* find(const char\* student\_id);  void print();  void clear();  };  #endif // AVL\_TREE\_H |  AVLTree.cpp  |  | | --- | | #include "AVLTree.h"  #include <algorithm>  #include <iostream>  Node::Node(const Student& student) : data(student), left(nullptr), right(nullptr), height(1) {}  AVLTree::AVLTree() : root(nullptr) {}  AVLTree::~AVLTree() {  clear(root);  }  int AVLTree::getHeight(Node\* node) {  if (!node) return 0;  return node->height;  }  int AVLTree::getBalanceFactor(Node\* node) {  if (!node) return 0;  return getHeight(node->left) - getHeight(node->right);  }  Node\* AVLTree::rightRotate(Node\* y) {  Node\* x = y->left;  Node\* T2 = x->right;  x->right = y;  y->left = T2;  y->height = std::max(getHeight(y->left), getHeight(y->right)) + 1;  x->height = std::max(getHeight(x->left), getHeight(x->right)) + 1;  return x;  }  Node\* AVLTree::leftRotate(Node\* x) {  Node\* y = x->right;  Node\* T2 = y->left;  y->left = x;  x->right = T2;  x->height = std::max(getHeight(x->left), getHeight(x->right)) + 1;  y->height = std::max(getHeight(y->left), getHeight(y->right)) + 1;  return y;  }  Node\* AVLTree::minValueNode(Node\* node) {  Node\* current = node;  while (current->left != nullptr)  current = current->left;  return current;  }  Node\* AVLTree::insert(Node\* node, const Student& student) {  if (!node) {  return new Node(student);  }  if (strcmp(student.student\_id, node->data.student\_id) < 0) {  node->left = insert(node->left, student);  } else if (strcmp(student.student\_id, node->data.student\_id) > 0) {  node->right = insert(node->right, student);  } else {  return node;  }  node->height = 1 + std::max(getHeight(node->left), getHeight(node->right));  int balance = getBalanceFactor(node);  if (balance > 1 && strcmp(student.student\_id, node->left->data.student\_id) < 0) {  return rightRotate(node);  }  if (balance < -1 && strcmp(student.student\_id, node->right->data.student\_id) > 0) {  return leftRotate(node);  }  if (balance > 1 && strcmp(student.student\_id, node->left->data.student\_id) > 0) {  node->left = leftRotate(node->left);  return rightRotate(node);  }  if (balance < -1 && strcmp(student.student\_id, node->right->data.student\_id) < 0) {  node->right = rightRotate(node->right);  return leftRotate(node);  }  return node;  }  void AVLTree::insert(const Student& student) {  root = insert(root, student);  }  Node\* AVLTree::deleteNode(Node\* node, const char\* student\_id) {  if (!node) return nullptr;  if (strcmp(student\_id, node->data.student\_id) < 0) {  node->left = deleteNode(node->left, student\_id);  } else if (strcmp(student\_id, node->data.student\_id) > 0) {  node->right = deleteNode(node->right, student\_id);  } else {  if (!node->left || !node->right) {  Node\* temp = node->left ? node->left : node->right;  if (!temp) {  temp = node;  node = nullptr;  } else {  \*node = \*temp;  }  delete temp;  } else {  Node\* temp = minValueNode(node->right);  node->data = temp->data;  node->right = deleteNode(node->right, temp->data.student\_id);  }  }  if (!node) return nullptr;  node->height = 1 + std::max(getHeight(node->left), getHeight(node->right));  int balance = getBalanceFactor(node);  if (balance > 1 && getBalanceFactor(node->left) >= 0) {  return rightRotate(node);  }  if (balance > 1 && getBalanceFactor(node->left) < 0) {  node->left = leftRotate(node->left);  return rightRotate(node);  }  if (balance < -1 && getBalanceFactor(node->right) <= 0) {  return leftRotate(node);  }  if (balance < -1 && getBalanceFactor(node->right) > 0) {  node->right = rightRotate(node->right);  return leftRotate(node);  }  return node;  }  void AVLTree::deleteStudent(const char\* student\_id) {  root = deleteNode(root, student\_id);  }  Student\* AVLTree::find(const char\* student\_id) {  Node\* result = find(root, student\_id);  return result ? &result->data : nullptr;  }  Node\* AVLTree::find(Node\* node, const char\* student\_id) {  if (!node) return nullptr;  if (strcmp(student\_id, node->data.student\_id) == 0) {  return node;  } else if (strcmp(student\_id, node->data.student\_id) < 0) {  return find(node->left, student\_id);  } else {  return find(node->right, student\_id);  }  }  void AVLTree::printTreeHelper(Node\* root, const std::string& prefix, bool isLeft) {  if (root != nullptr) {  std::cout << prefix;  std::cout << (isLeft ? "├──" : "└──" );  std::cout << root->data << std::endl;  printTreeHelper(root->left, prefix + (isLeft ? "│ " : " "), true);  printTreeHelper(root->right, prefix + (isLeft ? "│ " : " "), false);  }  }  void AVLTree::print() {  printTreeHelper(root, "", false);  }  void AVLTree::clear(Node\* node) {  if (node) {  clear(node->left);  clear(node->right);  delete node;  }  }  void AVLTree::clear() {  clear(root);  root = nullptr;  } | | Student.h  |  | | --- | | #ifndef STUDENT\_H  #define STUDENT\_H  #include <iostream>  #include <cstring>  struct Student {  char\* student\_id;  char\* full\_name;  int course;  char\* enrollment\_date;  char\* contact\_info;  Student();  Student(const char\* id, const char\* name, int crs, const char\* date, const char\* contact);  ~Student();  Student(const Student& other);  Student& operator=(const Student& other);  friend std::ostream& operator<<(std::ostream& os, const Student& student);  };  #endif // STUDENT\_H |  Student.cpp  |  | | --- | | #include "Student.h"  Student::Student() : student\_id(nullptr), full\_name(nullptr), course(0), enrollment\_date(nullptr), contact\_info(nullptr) {}  Student::Student(const char\* id, const char\* name, int crs, const char\* date, const char\* contact) : course(crs) {  student\_id = new char[strlen(id) + 1];  strcpy(student\_id, id);  full\_name = new char[strlen(name) + 1];  strcpy(full\_name, name);  enrollment\_date = new char[strlen(date) + 1];  strcpy(enrollment\_date, date);  contact\_info = new char[strlen(contact) + 1];  strcpy(contact\_info, contact);  }  Student::~Student() {  delete[] student\_id;  delete[] full\_name;  delete[] enrollment\_date;  delete[] contact\_info;  }  Student::Student(const Student& other) : course(other.course) {  student\_id = new char[strlen(other.student\_id) + 1];  strcpy(student\_id, other.student\_id);  full\_name = new char[strlen(other.full\_name) + 1];  strcpy(full\_name, other.full\_name);  enrollment\_date = new char[strlen(other.enrollment\_date) + 1];  strcpy(enrollment\_date, other.enrollment\_date);  contact\_info = new char[strlen(other.contact\_info) + 1];  strcpy(contact\_info, other.contact\_info);  }  Student& Student::operator=(const Student& other) {  if (this != &other) {  delete[] student\_id;  delete[] full\_name;  delete[] enrollment\_date;  delete[] contact\_info;  student\_id = new char[strlen(other.student\_id) + 1];  strcpy(student\_id, other.student\_id);  full\_name = new char[strlen(other.full\_name) + 1];  strcpy(full\_name, other.full\_name);  enrollment\_date = new char[strlen(other.enrollment\_date) + 1];  strcpy(enrollment\_date, other.enrollment\_date);  contact\_info = new char[strlen(other.contact\_info) + 1];  strcpy(contact\_info, other.contact\_info);  course = other.course;  }  return \*this;  }  std::ostream& operator<<(std::ostream& os, const Student& student) {  os << "ID: " << student.student\_id << ", ФИО: " << student.full\_name << ", Курс: " << student.course  << ", Дата зачисления: " << student.enrollment\_date << ", Контакты: " << student.contact\_info;  return os;  } |  main.cpp  |  | | --- | | #include "AVLTree.h"  #include <iostream>  #include <string>  void addStudent(AVLTree& tree) {  std::string student\_id, full\_name, enrollment\_date, contact\_info;  int course;  std::cin.ignore();  std::cout << "Введите номер студенческого билета: ";  std::getline(std::cin, student\_id);  std::cout << "Введите ФИО студента: ";  std::getline(std::cin, full\_name);  std::cout << "Введите курс: ";  std::cin >> course;  std::cin.ignore();  std::cout << "Введите дату зачисления (dd.mm.yyyy): ";  std::getline(std::cin, enrollment\_date);  std::cout << "Введите контактную информацию: ";  std::getline(std::cin, contact\_info);  Student new\_student(student\_id.c\_str(), full\_name.c\_str(), course, enrollment\_date.c\_str(), contact\_info.c\_str());  tree.insert(new\_student);  }  void findStudent(AVLTree& tree) {  std::string student\_id;  std::cout << "Введите номер студенческого билета для поиска: ";  std::cin >> student\_id;  Student\* found = tree.find(student\_id.c\_str());  if (found) {  std::cout << "Студент найден: " << \*found << std::endl;  } else {  std::cout << "Студент с таким номером билета не найден." << std::endl;  }  }  void deleteStudent(AVLTree& tree) {  std::string student\_id;  std::cout << "Введите номер студенческого билета для удаления: ";  std::cin >> student\_id;  tree.deleteStudent(student\_id.c\_str());  std::cout << "Студент удален." << std::endl;  }  int main() {  AVLTree tree;  bool running = true;  int choice;  while (running) {  std::cout << "\nМеню:\n";  std::cout << "1. Добавить студента\n";  std::cout << "2. Вывод информации о всех студентах\n";  std::cout << "3. Поиск студента\n";  std::cout << "4. Удаление студента\n";  std::cout << "5. Очистить все данные\n";  std::cout << "6. Выход\n";  std::cout << "Выберите действие: ";  std::cin >> choice;  switch (choice) {  case 1:  addStudent(tree);  break;  case 2:  tree.print();  break;  case 3:  findStudent(tree);  break;  case 4:  deleteStudent(tree);  break;  case 5:  tree.clear();  std::cout << "Все данные очищены." << std::endl;  break;  case 6:  running = false;  break;  default:  std::cout << "Неверный ввод, попробуйте еще раз.\n";  }  }  return 0;  } | |