****

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«МИРЭА – Российский технологический университет»**

# РТУ МИРЭА

Отчет по выполнению практического задания №2.4

**Тема:** Бинарное дерево поиска. AVL дерево

Дисциплина: **«**СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ**»**

Выполнил студент Кузнецов А. А.

группа ИКБО-01-21

**Москва 2022**

**Цель работы:** составить программу создания двоичного дерева поиска и реализовать процедуры для работы с деревом согласно варианту.

**Задание:**

1. Процедуры оформить в виде самостоятельных режимов работы созданного дерева. Выбор режимов производить с помощью пользовательского (иерархического ниспадающего) меню.
2. Провести полное тестирование программы на дереве размером n=10 элементов, сформированном вводом с клавиатуры. Тест-примеры определить самостоятельно. Результаты тестирования в виде скриншотов экранов включить в отчет по выполненной работе.
3. Сделать выводы о проделанной работе, основанные на полученных результатах.
4. Оформить отчет с подробным описанием созданного дерева, принципов программной реализации алгоритмов работы с деревом, описанием текста исходного кода и проведенного тестирования программы.

**Индивидуальный вариант:**

Вариант: 22

Тип значения узла: Строка – имя

Тип дерева: Бинарное дерево поиска

Реализовать алгоритмы:

* Вставка элемента
* Прямой обход
* Симметричный обход
* Найти длину пути от корня до заданного значения
* Найти высоту дерева

**Математическая модель решения**

**Структура элементов дерева.** Для выполнения задания индивидуального варианта создадим структуру индивидуального элемента дерева (Нода дерева) с полями:

* Имени (string)
* Высоты элемента (int)
* Указатель на родителя и потомков: левый и правый (Node\*)
* Конструктор нода с параметром имени (string key)

**Код структуры Node:**

|  |
| --- |
| // ноды бинарного дерева  struct Node {  string name;  int height;  Node\* left;  Node\* right;  Node\* parent;  Node(string key) : name(key), height(0), left(NULL), right(NULL), parent(NULL) {};  }; |

**Алгоритм вставки.** Также важно учесть алгоритм вставки элементов – Ноды вставляются путем сравнения поля имени нода (по алфавиту): если имя текущего неода выше по алфавиту, то программа вставляет нод дерева с этим именем в левого потомка, иначе в правого потомка. В случае, если нод-потомок уже занят, то программа спускается ниже и сравнивает поле имени уже с нодом-потомком и так далее, пока не будет найдено свободное место для нода.

**Код вставки элемента в дерево:**

|  |
| --- |
| // проходим по всему дереву и в алфавитном порядке вставляем новый нод  void insert(Node\* root, Node\* node) {  while (root) {  if (node->name < root->name) {  if (root->left) {  root = root->left; // если нод (левый) занят, перемещаемся к нему и начинаем заново  node->height++;  }  else {  node->parent = root; // вствляем нод  root->left = node;  break;  }  }  else if (node->name > root->name) {  if (root->right) {  root = root->right; // если нод (правый) занят, перемещаемся к нему и начинаем заново  node->height++;  }  else {  node->parent = root; // вствляем нод  root->right = node;  break;  }  }  }  } |

**Прямой и симметричный проходы по дереву.** Проходы по дереву сделаем рекурсивно. В прямом программа будет рекурсивно заходить в левых потомков, пока они есть, потом выводить нод и далее заходить в правых потомков и так далее. В симметричном же проходе программа будет сначала выводить нод-родитель, а потом рекурсивно заходить в левых потомков и далее в правых.

**Код реализация прямого и симметричного проходов:**

|  |
| --- |
| // прямой обход  void inorderTraversal(Node\* x) {  if (x) { // пока есть нод  inorderTraversal(x->left); // заходим в рекурсию до самого левого нода  cout << x->name << " "; // выводим нод  inorderTraversal(x->right); // заходим в рекурсию в правый  }  else return;  }  // симметричный обход  void symmetricalTraversal(Node\* x) {  if (x) { // пока есть нод  cout << x->name << " "; // выводим текущий нод рекурсии  symmetricalTraversal(x->left); // рекурсионно заходим в левый, выводим его и далее  symmetricalTraversal(x->right); // рекурсионно заходим в правые оставшиеся  }  else return;  } |

**Высота дерева.** Для нахождения максимальной высоты дерева воспользуемся так же рекурсией для прохода по нодам дерева и с помощью сравнения полей высот нодов (height), ищем максимальную и возвращаем ее.

**Код нахождения высоты дерева:**

|  |
| --- |
| // нахождение выстоты дерева  int treeHeight(Node\* x) {  int maxH = -1;  int result;  if (x) { // пока есть нод  if (maxH < x->height)  maxH = x->height;  result = treeHeight(x->left); // рекурсионно заходим в левые  if (maxH < result)  maxH = result;  result = treeHeight(x->right); // рекурсионно заходим в правые оставшиеся  if (maxH < result)  maxH = result;  }  return maxH;  } |

**Поиск длины пути от корня до заданного элемента.** Чтобы найти глубину/высоту элемента, опять же будем делать рекурсивный проход по элементам дерева и сравнивать их с именем, переданным в параметре функции. Если элемент с нужным именем будет найден, то вернем его значение поля высоты (height).

**Код нахождение глубины элемента по имени:**

|  |
| --- |
| // находжение глубины нода дерева по имени  int searchElementsDeep(Node\* x, string key) {  int res = -1;  int temp = -1;  if (x) { // пока есть нод  if (key == x->name)  res = x->height;  if ((temp = searchElementsDeep(x->left, key)) != -1)  res = temp; // рекурсионно заходим в левые  if ((temp = searchElementsDeep(x->right, key)) != -1)  res = temp; // рекурсионно заходим в левые  }  return res;  } |

**Результаты тестирования**

При запуске программы, она предлагает нам ввести начальное количество элементов в дереве. Введем 5 элементов и имена. (см. Рис 1)

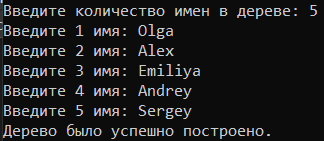


Рисунок 1. Построение начального дерева.

Нарисуем граф дерева, которое должно было построиться с именами из примера в рисунке 1. (см. Рис 2)

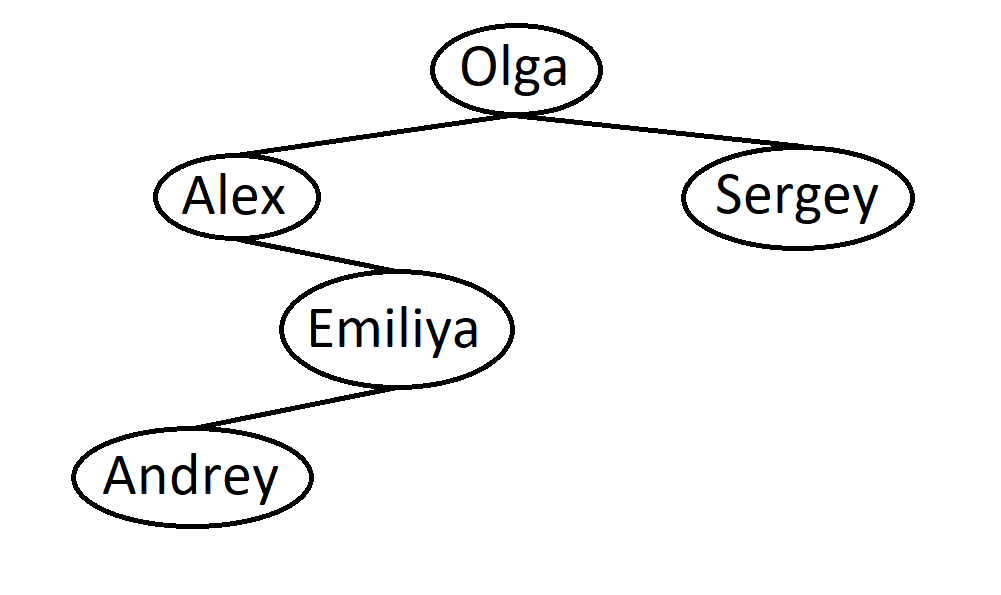


Рисунок 2. Граф, который должен получится из имен в примере.

После того, как программа сообщила нам об успешной постройке дерева, она предлагает нам на выбор несколько команд для работы с деревом.

**Прямой обход.** Проверим работу второй команды (Прямой обход) и корректность построения дерева. При прямом обходе программа должна вывести имена в алфавитном порядке. (см. Рис 3)

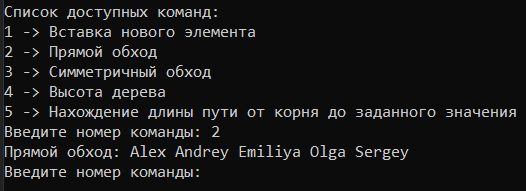


Рисунок 3. Проверка прямого обхода с выводом элементов дерева

Вывод корректный. При прямом обходе имена должны были быть выведены в порядке как на рисунке ниже и было описано в математической модели решения ранее - слева направо. (см. Рис 4)

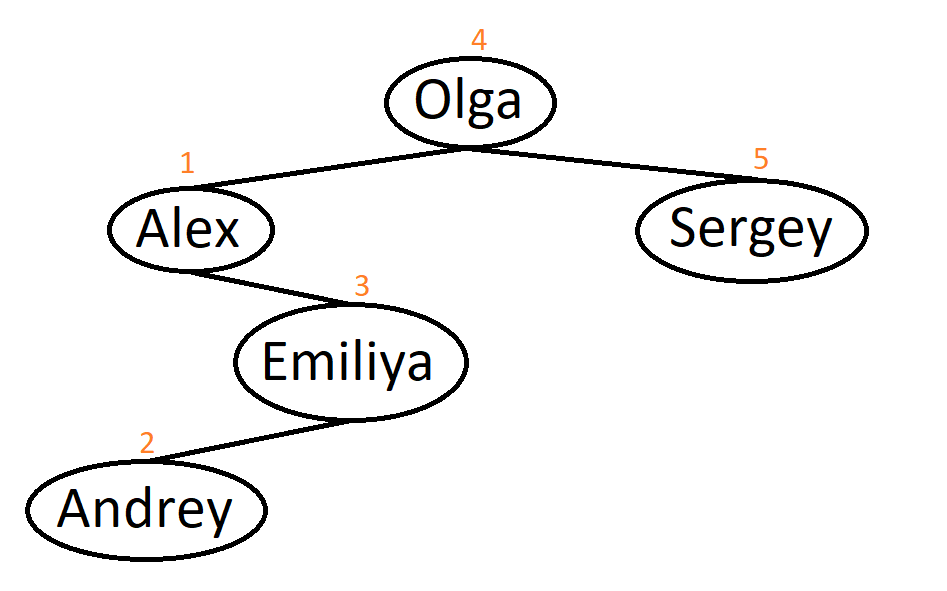


Рисунок 4. Порядок вывода при прямом обходе в соответствии с математической моделью решения

**Симметричный обход.** Вывод программы (см. Рис 5). Правильный порядок в соответствии с математической моделью решения – родитель, левые потомки, правые потомки. (см. Рис 6).



Рисунок 5. Вывод элементов в соответствии симметричного обхода

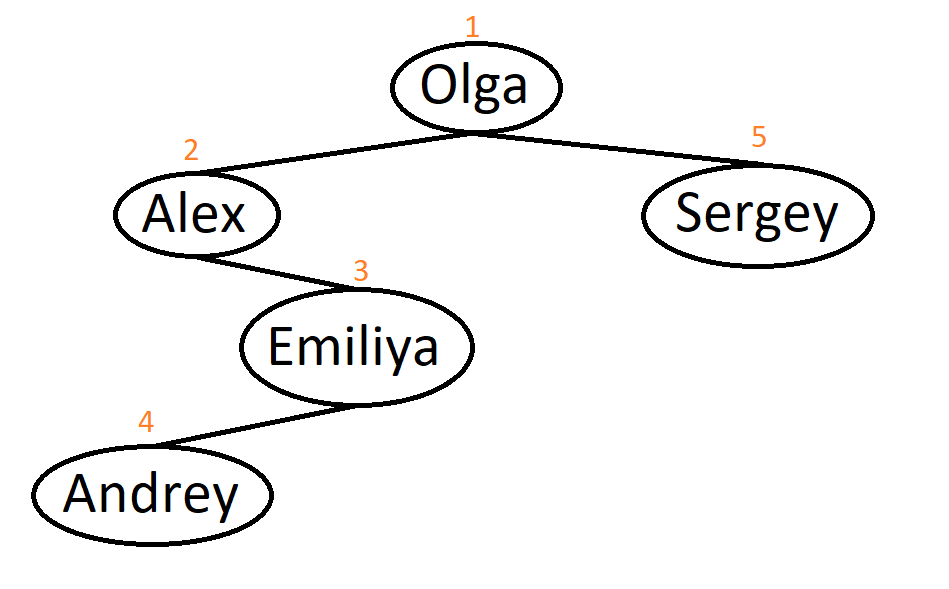


Рисунок 6. Правильный порядок вывода элементов при симметричном обходе в соответствии с математической моделью решения

**Вставка нового элемента. Прямой и симметричный обходы с новым элементом.** Вставим новый элемент и проверим корректность работы обходов. Вставим новый элемент при помощи команды №1 и проверим обходы. (см. Рис 7)

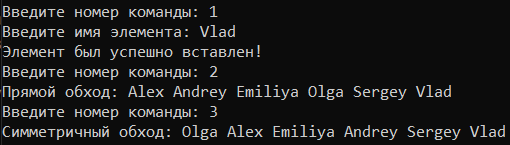


Рисунок 7. Результат выполнения вставки нового элемента, прямого и симметричного обходов.

Правильные порядки обходов. (см. Рис 8)

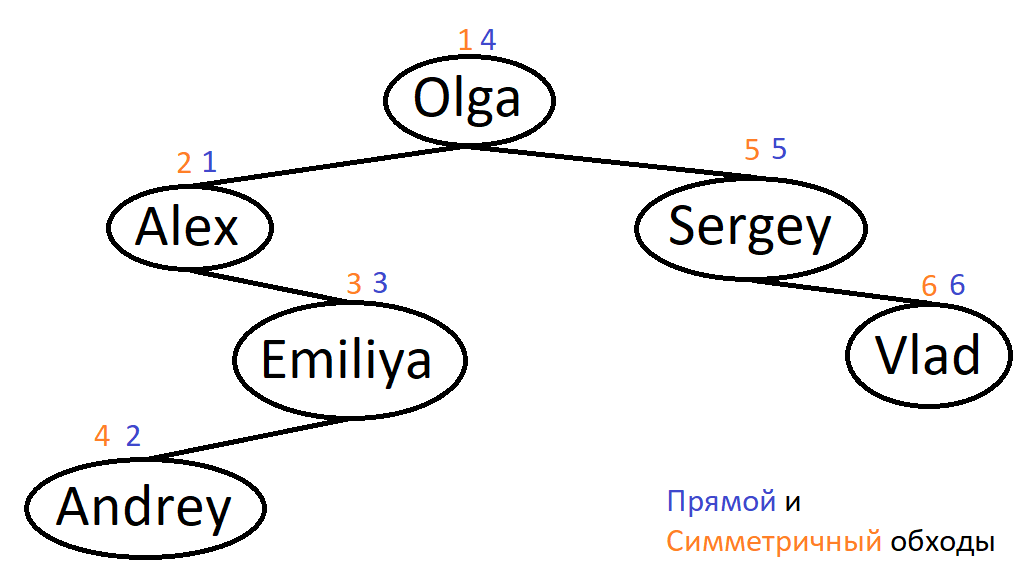


Рисунок 8. Правильный порядок вывода прямого и симметричного обходов.

**Высота дерева.** Проверим корректность выполнения команды №4 – вывод высоты дерева (см. Рис 9). Правильная высота дерева (см. рис 10).

****

Рисунок 9. Вывод высоты дерева.

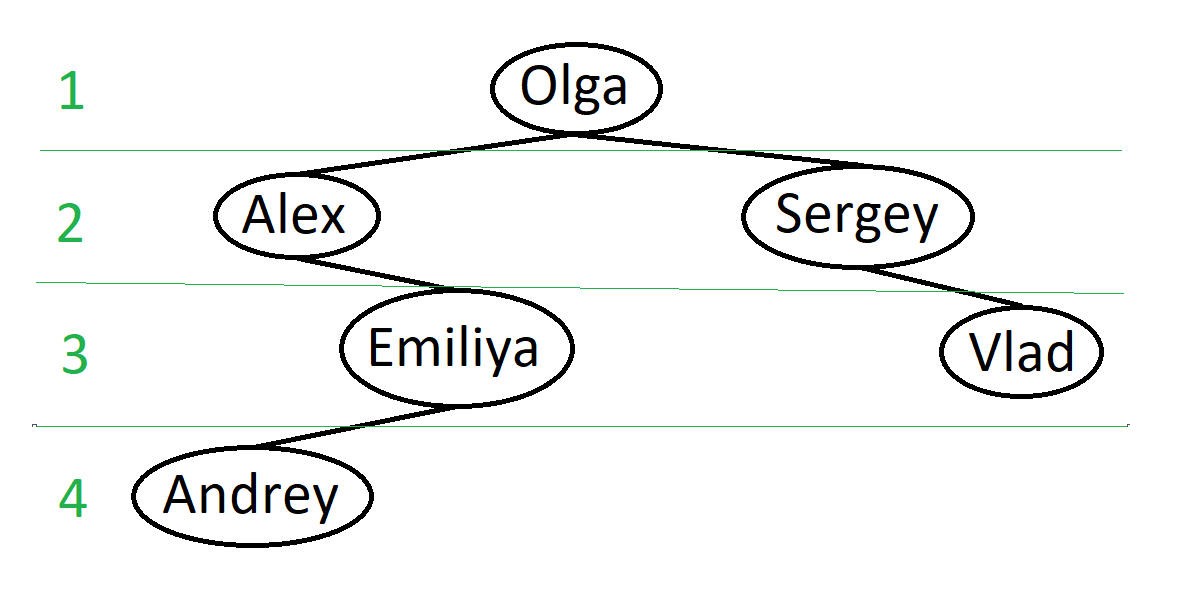


Рисунок 10. Реальная высота дерева

**Нахождения расстояния элемента от корневого.** Проверим корректность работы команды №5 – расстояние элемента от корневого (см. Рис 11). Расстояния на дереве (см. Рис 12).

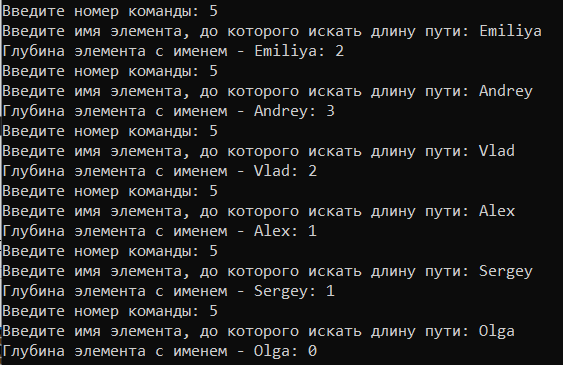


Рисунок 11. Вывод расстояния элементов от корневого

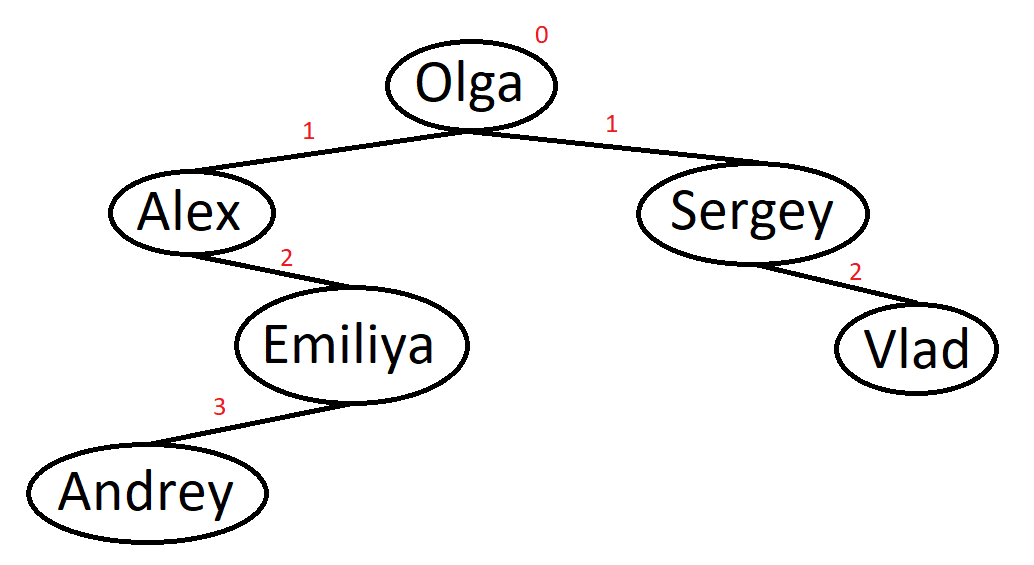


Рисунок 12. Реальные расстояния элементов.

**Вывод.** Все результаты тестирования показали корректность работы программы.

**Задание 1**

Программа предлагает выбрать номер задания (см. Рис 1). Введем “1” и текст со знаками препинания и разными регистрами букв для проверки задания (см. Рис. 2).

****

Рисунок 1. Ввод номера задания.

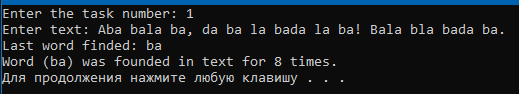


Рисунок 2. Ввод текста и проверка выполнения.

По результатам тестирования подстрока (последнее слово – ba) было найдено верное количество раз (8 раз).

**Задание 2**

Теперь выберем второе задание и скопируем 7 раз предложение. В качестве подстроки введем часть этого предложения. (см. Рис. 3).

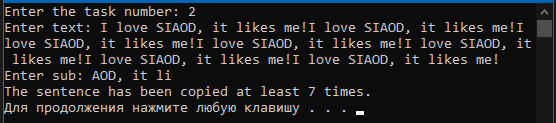
****

Рисунок 3. Проверка второго задания.

По результатам тестирования программа определила, что предложение было скопировано как минимум 7 раз, что является корректным значением.

**Полный код программы**

|  |
| --- |
| #include <iostream>  #include <string>  #include "locale.h"  using namespace std;  // ноды бинарного дерева  struct Node {  string name;  int height;  Node\* left;  Node\* right;  Node\* parent;  Node(string key) : name(key), height(0), left(NULL), right(NULL), parent(NULL) {};  };  // проходим по всему дереву и в алфавитном порядке вставляем новый нод  void insert(Node\* root, Node\* node) {  while (root) {  if (node->name < root->name) {  if (root->left) {  root = root->left; // если нод (левый) занят, перемещаемся к нему и начинаем заново  node->height++;  }  else {  node->parent = root; // вствляем нод  root->left = node;  break;  }  }  else if (node->name > root->name) {  if (root->right) {  root = root->right; // если нод (правый) занят, перемещаемся к нему и начинаем заново  node->height++;  }  else {  node->parent = root; // вствляем нод  root->right = node;  break;  }  }  }  }  // прямой обход  void inorderTraversal(Node\* x) {  if (x) { // пока есть нод  inorderTraversal(x->left); // заходим в рекурсию до самого левого нода  cout << x->name << " "; // выводим нод  inorderTraversal(x->right); // заходим в рекурсию в правый  }  else return;  }  // симметричный обход  void symmetricalTraversal(Node\* x) {  if (x) { // пока есть нод  cout << x->name << " "; // выводим текущий нод рекурсии  symmetricalTraversal(x->left); // рекурсионно заходим в левый, выводим его и далее  symmetricalTraversal(x->right); // рекурсионно заходим в правые оставшиеся  }  else return;  }  // нахождение выстоты дерева  int treeHeight(Node\* x) {  int maxH = -1;  int result;  if (x) { // пока есть нод  if (maxH < x->height)  maxH = x->height;  result = treeHeight(x->left); // рекурсионно заходим в левые  if (maxH < result)  maxH = result;  result = treeHeight(x->right); // рекурсионно заходим в правые оставшиеся  if (maxH < result)  maxH = result;  }  return maxH;  }  // находжение глубины нода дерева по имени  int searchElementsDeep(Node\* x, string key) {  int res = -1;  int temp = -1;  if (x) { // пока есть нод  if (key == x->name)  res = x->height;  if ((temp = searchElementsDeep(x->left, key)) != -1)  res = temp; // рекурсионно заходим в левые  if ((temp = searchElementsDeep(x->right, key)) != -1)  res = temp; // рекурсионно заходим в левые  }  return res;  }  int main()  {  setlocale(LC\_ALL, "Russian");  cout << "Введите количество имен в дереве: ";  int n;  Node\* root = new Node("");  cin >> n;  for (int i = 0; i < n; i++) {  cout << "Введите " << i + 1 << " имя: ";  string town;  cin >> town;  insert(root, new Node(town));  }  cout << "Дерево было успешно построено.\n\n";  cout << "Список доступных команд:\n"  << "1 -> Вставка нового элемента\n"  << "2 -> Прямой обход\n"  << "3 -> Симметричный обход\n"  << "4 -> Высота дерева\n"  << "5 -> Нахождение длины пути от корня до заданного значения\n"  << "Введите номер команды: ";  int command;  string node;  string key;  while (true) {  cin >> command;  switch (command)  {  case 1:  cout << "Введите имя элемента: ";  cin >> node;  insert(root, new Node(node));  cout << "Элемент был успешно вставлен!\n";  break;  case 2:  cout << "Прямой обход:";  inorderTraversal(root);  cout << endl;  break;  case 3:  cout << "Симметричный обход:";  symmetricalTraversal(root);  cout << endl;  break;  case 4:  cout << "Высота текущего состояния дерева: ";  cout << treeHeight(root) << endl;  break;  case 5:  cout << "Введите имя элемента, до которого искать длину пути: ";  cin >> key;  cout << "Глубина элемента с именем - " << key << ": ";  cout << searchElementsDeep(root, key) << endl;  break;  default:  cout << "Неизвестная команда, попоробуйте еще раз.\n";  cin >> command;  break;  }  cout << "Введите номер команды: ";  }  } |

**ВЫВОДЫ**

В результате выполнения данной работы мной были освоены навыки программного построения бинарных деревьев и работы с ними. Была реализована программа для построения бинарного дерева и функции над его редактированием и работы.