# Лабораторная работа №5

Выполнил: Лосев Данил

**Задания:**

1. Реализуйте код класса **Tree**, с методами:

* *Создание дерева* идобавление новых звеньев в паре с методом *Поиск со вставкой*
* *Вывод дерева на консоль*
* *Очистки памяти от дерева*

1. Добавьте самостоятельно в код класса:

* *Обходы дерева*
* *Вычисление высоты дерева*

1. Используйте для реализации удобный для Вас редактор кода на языке С++.
2. Оформить отчет в электронном виде.

|  |
| --- |
| #include <cstddef>  #include <cstdlib>  #include <iostream>  // Узел дерева, который хранит ключ (Key), количество его добавлений (Count),  // а также указатели на левое (Left) и правое (Right) поддеревья.  struct Node  {  int Key; // Ключ узла  int Count; // Количество добавлений данного ключа  Node \*Left; // Указатель на левое поддерево  Node \*Right; // Указатель на правое поддерево  };  // Класс для работы с бинарным деревом поиска.  class TREE  {  private:  Node \*Tree; // Указатель на корень дерева  // Функция для поиска места для нового элемента в дереве  void Search(int, Node \*\*);  public:  // Конструктор, инициализирующий дерево пустым  TREE() : Tree(NULL)  {  }  // Возвращает указатель на корень дерева  Node \*\*GetTree()  {  return &Tree;  }  // Построение дерева с элементами от пользователя  void BuildTree();  // Очистка дерева, удаление всех узлов  void CleanTree(Node \*\*);  // Вывод дерева на экран  void PrintTree(Node \*\*, int = 0);  // Прямой порядок обхода дерева (PreOrder)  void PreOrder(Node \*\*);  // Симметричный порядок обхода дерева (InOrder)  void InOrder(Node \*\*);  // Обратный порядок обхода дерева (PostOrder)  void PostOrder(Node \*\*);  // Определение высоты дерева  int Height(Node \*\*);  };  int main()  {  TREE myTree; // Создание объекта дерева  myTree.BuildTree(); // Построение дерева из данных пользователя  std::cout << "\nPrinting the tree: " << std::endl;  myTree.PrintTree(myTree.GetTree()); // Вывод структуры дерева  std::cout << "\nPre order: ";  myTree.PreOrder(myTree.GetTree()); // Прямой обход дерева  std::cout << "\nIn order: ";  myTree.InOrder(myTree.GetTree()); // Симметричный обход дерева  std::cout << "\nPost order: ";  myTree.PostOrder(myTree.GetTree()); // Обратный обход дерева  // Вывод высоты дерева  std::cout << "\nHeight of tree: " << myTree.Height(myTree.GetTree()) << std::endl;  myTree.CleanTree(myTree.GetTree()); // Очистка дерева от всех узлов  std::cout << "The tree is cleaned up" << std::endl;  return 0;  }  // Функция для построения дерева на основе пользовательского ввода  void TREE::BuildTree()  {  int element;  std::cout << "Enter the keys of tree`s tops (0 = end): " << std::endl;  std::cin >> element; // Ввод первого элемента  while (element != 0) // Продолжаем, пока не введен 0  {  Search(element, &Tree); // Добавление элемента в дерево  std::cin >> element; // Ввод следующего элемента  }  }  // Поиск места для нового элемента в дереве. Если элемент уже существует, увеличивается его счетчик.  void TREE::Search(int fElement, Node \*\*fTree)  {  if (\*fTree == NULL)  {  \*fTree = new (Node); // Создание нового узла  (\*\*fTree).Key = fElement; // Присвоение ключа  (\*\*fTree).Count = 1; // Установка начального значения счетчика  (\*\*fTree).Left = NULL; // Левое поддерево пусто  (\*\*fTree).Right = NULL; // Правое поддерево пусто  }  else  {  if (fElement < (\*\*fTree).Key) // Если новый элемент меньше, идем влево  {  Search(fElement, &(\*\*fTree).Left);  }  else if (fElement > (\*\*fTree).Key) // Если больше, идем вправо  {  Search(fElement, &(\*\*fTree).Right);  }  else  {  (\*\*fTree).Count++; // Если элемент найден, увеличиваем его счетчик  }  }  }  // Вывод структуры дерева на экран с отступами для отображения уровней  void TREE::PrintTree(Node \*\*fTree, int index)  {  int i;  if (\*fTree != NULL) // Если узел существует  {  PrintTree(&(\*\*fTree).Right, index + 1); // Вывод правого поддерева  for (i = 1; i <= index; i++) // Добавление отступов для визуализации уровня  {  std::cout << " ";  }  std::cout << (\*\*fTree).Key << std::endl; // Вывод ключа узла  PrintTree(&(\*\*fTree).Left, index + 1); // Вывод левого поддерева  }  }  // Очистка дерева: рекурсивно удаляются все узлы дерева  void TREE::CleanTree(Node \*\*fTree)  {  if (\*fTree != NULL) // Если узел существует  {  CleanTree(&(\*\*fTree).Left); // Очистка левого поддерева  CleanTree(&(\*\*fTree).Right); // Очистка правого поддерева  delete \*fTree; // Удаление текущего узла  }  }  // Прямой (PreOrder) обход дерева: корень, левое поддерево, правое поддерево  void TREE::PreOrder(Node \*\*fTree)  {  if (\*fTree != NULL)  {  std::cout << (\*\*fTree).Key << " "; // Вывод ключа узла  PreOrder(&(\*\*fTree).Left); // Обход левого поддерева  PreOrder(&(\*\*fTree).Right); // Обход правого поддерева  }  }  // Симметричный (InOrder) обход дерева: левое поддерево, корень, правое поддерево  void TREE::InOrder(Node \*\*fTree)  {  if (\*fTree != NULL)  {  InOrder(&(\*\*fTree).Left); // Обход левого поддерева  std::cout << (\*\*fTree).Key << " "; // Вывод ключа узла  InOrder(&(\*\*fTree).Right); // Обход правого поддерева  }  }  // Обратный (PostOrder) обход дерева: левое поддерево, правое поддерево, корень  void TREE::PostOrder(Node \*\*fTree)  {  if (\*fTree != NULL)  {  PostOrder(&(\*\*fTree).Left); // Обход левого поддерева  PostOrder(&(\*\*fTree).Right); // Обход правого поддерева  std::cout << (\*\*fTree).Key << " "; // Вывод ключа узла  }  }  // Определение высоты дерева  int TREE::Height(Node \*\*fTree)  {  int heightRight, heightLeft;  if (\*fTree == NULL)  {  return -1; // Пустое дерево имеет высоту -1  }  else  {  heightLeft = Height(&(\*\*fTree).Left); // Высота левого поддерева  heightRight = Height(&(\*\*fTree).Right); // Высота правого поддерева  if (heightLeft > heightRight) // Возвращается максимальная высота  {  return (1 + heightLeft);  }  else  {  return (1 + heightRight);  }  }  } |

