**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра АПУ**

**ОТЧЕТ**

**по лабораторной работе №6**

**по дисциплине «Программирование»**

| Студент гр. 3391 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Николаев В.Ю. |
| --- | --- | --- |
| Преподаватель | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Писарев А.С. |

Санкт-Петербург

2024 г.

# Исходная формулировка

Реализовать программу построения и обработки бинарного дерева поиска. Дерево заполняется вводимыми с клавиатуры числами (признак окончания входной последовательности – ввод числа 0). После этого должен поддерживаться быстрый поиск задаваемых с клавиатуры чисел в построенном дереве. По окончании операций поиска программа должна очистить память, занимаемую древесной структурой данных.

# Ход работы программы

Программа начинается с включения заголовочного файла "BinarySearchTree.h", который содержит определение класса BinarySearchTree.

| 1 | #include "BinarySearchTree.h" |
| --- | --- |

В функции main создается объект tree класса BinarySearchTree.

| 5 | BinarySearchTree tree; |
| --- | --- |

Затем программа выводит сообщение, предлагая пользователю ввести числа для вставки в дерево. Ввод продолжается до тех пор, пока пользователь не введет 0.

| 7  8  9  10  11  12  13  14  15 | std::cout << "Enter numbers to insert into the tree (0 to stop): ";  int tmp;  do  {  std::cin >> tmp;  if (tmp != 0)  tree.insert(tmp);  }  while (tmp != 0); |
| --- | --- |

После этого программа выводит содержимое дерева.

| 17  18 | std::cout << "Tree: " << std::endl;  tree.print(); |
| --- | --- |

Затем программа предлагает пользователю ввести число для поиска в дереве.

| 20  21 | std::cout << "Enter a number to search for: ";  std::cin >> tmp; |
| --- | --- |

Программа ищет введенное число в дереве и выводит результат поиска.

| 23  24  25  26 | if (tree.find(tmp))  std::cout << "Found\n";  else  std::cout << "Not found\n"; |
| --- | --- |

# 

# Текст программы

| Main.cpp | |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | #include "BinarySearchTree.h"  int main()  {  BinarySearchTree tree;  std::cout << "Enter numbers to insert into the tree (0 to stop): ";  int tmp;  do  {  std::cin >> tmp;  if (tmp != 0)  tree.insert(tmp);  }  while (tmp != 0);  std::cout << "Tree: " << std::endl;  tree.print();  std::cout << "Enter a number to search for: ";  std::cin >> tmp;  if (tree.find(tmp))  std::cout << "Found\n";  else  std::cout << "Not found\n";  return 0;  } |
| BinarySearchTree.h | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142 | #pragma once  #include <iostream>  class BinarySearchTree  {  private:  struct Node  {  int data;  Node\* left;  Node\* right;  Node(int data, Node\* left, Node\* right):  data(data),  left(left),  right(right)  {}  ~Node()  {  delete left;  delete right;  }  };  Node\* root;  void insert(Node\*& node, int data)  {  if (node == nullptr)  {  node = new Node(data, nullptr, nullptr);  }  else if (data < node->data)  insert(node->left, data);  else  insert(node->right, data);    }  bool find(Node\* node, int data)  {  if (node == nullptr)  return false;  else if (data == node->data)  return true;  else if (data < node->data)  return find(node->left, data);  else  return find(node->right, data);  }  void printHelper(int level, Node\* node, int mode, char\* modes) const  {  if (node == nullptr)  return;  modes[level + 1] = 'r';  printHelper(level + 1, node->right, 1, modes);  modes[level] = (mode == 0) ? 'm' : (mode == 1) ? 'v' : 'l';  for (int i = 0; i < level; i++)  if (modes[i] == 'v')  std::cout << "│ ";  else  std::cout << " ";  switch (mode)  {  case 1:  std::cout << "┌─";  break;  case 0:  std::cout << "├─";  break;    case -1:  std::cout << "└─";  break;  }  std::cout << node->data << std::endl;  modes[level + 1] = 'v';  printHelper(level + 1, node->left, -1, modes);  }  int height(Node\* node) const  {  if (node == nullptr)  return 0;  else  {  int leftHeight = height(node->left);  int rightHeight = height(node->right);  return 1 + ((leftHeight > rightHeight) ? leftHeight : rightHeight);  }  }  public:  BinarySearchTree()  {  root = nullptr;  }  void insert(int data)  {  insert(root, data);  }  bool find(int data)  {  return find(root, data);  }  void print() const  {  int h = height(root);  char \*modes = new char[h + 1];  modes[0] = 'm';  for (int i = 1; i < h + 1; i++)  modes[i] = 'v';  printHelper(0, root, 0, modes);  delete[] modes;  }  void clear()  {  delete root;  root = nullptr;  }  ~BinarySearchTree()  {  delete root;  }  }; |

# 

# Результаты работы программы