**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра АПУ**

**ОТЧЕТ**

**по лабораторной работе №7**

**по дисциплине «Программирование»**

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

Санкт-Петербург

2024 г.

# Исходная формулировка

Расширить функциональность программы из лабораторной работы №6 путем ее дополнения функцией вывода элементов построенного дерева на экран:

1. в префиксном порядке;
2. в инфиксном порядке;
3. в постфиксном порядке.

Дополнить программу вводом соответствующего меню, позволяющего выбирать номера пунктов задания.

# Ход работы программы

После завершения уже описанной в 6 лабораторной работе части работы программы, программа предлагает пользователю выбрать порядок обхода дерева (предварительный, внутренний или последующий) и выводит дерево в выбранном порядке.

| 28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50 | std::cout << "1. Pre-order" << std::endl;  std::cout << "2. In-order" << std::endl;  std::cout << "3. Post-order" << std::endl;  std::cout << "Enter a number to print the tree in the desired order: ";  std::cin >> tmp;  switch (tmp)  {  case 1:  tree.preOrder();  break;  case 2:  tree.inOrder();  break;  case 3:  tree.postOrder();  break;    default:  break;  }  std::cout << std::endl; |
| --- | --- |

# 

# 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  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53 | #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";  std::cout << "1. Pre-order" << std::endl;  std::cout << "2. In-order" << std::endl;  std::cout << "3. Post-order" << std::endl;  std::cout << "Enter a number to print the tree in the desired order: ";  std::cin >> tmp;  switch (tmp)  {  case 1:  tree.preOrder();  break;  case 2:  tree.inOrder();  break;  case 3:  tree.postOrder();  break;    default:  break;  }  std::cout << std::endl;  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  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198 | #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);  }  }  void preOrder(Node\* node) const  {  if (node == nullptr)  return;  std::cout << node->data << " ";  preOrder(node->left);  preOrder(node->right);  }  void inOrder(Node\* node) const  {  if (node == nullptr)  return;  inOrder(node->left);  std::cout << node->data << " ";  inOrder(node->right);  }  void postOrder(Node\* node) const  {  if (node == nullptr)  return;  postOrder(node->left);  postOrder(node->right);  std::cout << node->data << " ";  }  void clear(Node\* node)  {  if (node == nullptr)  return;  clear(node->left);  clear(node->right);  delete node;  }  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 preOrder() const  {  preOrder(root);  }  void inOrder() const  {  inOrder(root);  }  void postOrder() const  {  postOrder(root);  }  void clear()  {  clear(root);  root = nullptr;  }  ~BinarySearchTree()  {  delete root;  }  }; |

# 

# Результаты работы программы

|  |  |
| --- | --- |
|  | |