**Отчёт по лабораторной работе №2**

**Бинарные деревья**

Задание:

Написать функцию формирования бинарного дерева, состоящего из целых чисел. Для представления дерева использовать динамические структуры данных. Количество элементов дерева, а также его вид задаются случайным образом. Произвести вывод элементов дерева тремя видами обхода. Используя информацию о выведенном дереве изобразить структуру одного из построенных деревьев в отчете.

14. Написать рекурсивную функцию, которая выбирает все вершины с отрицательными значениями и строит из них новое дерево.

Код программы:

|  |  |  |
| --- | --- | --- |
| Source.cpp | bin\_tree.cpp | bin\_tree.h |
| #include <iostream>  #include "bin\_tree.h"  void copy\_tree(BinTree\* tree, BinTree\* neg\_tree, BinTree::Node\* node) {  if (node != NULL) {  if (node->data < 0)  neg\_tree->add(node->data);  copy\_tree(tree, neg\_tree, node->left);  copy\_tree(tree, neg\_tree, node->right);  }  }  int main() {  srand(578);  std::cout << "Tree 1:" << std::endl;  int size = 5 + rand() % 15;  BinTree tree = BinTree(size);  tree.generate();  std::cout << " - Direct pass: ";  tree.print(BinTree::P\_DIRECT);  std::cout << " - Parity pass: " ;  tree.print(BinTree::P\_PARITY);  std::cout << " - Reverse pass: " ;  tree.print(BinTree::P\_REVERSE);  std::cout << std::endl;  std::cout << "Tree 2:" << std::endl;  BinTree neg\_tree = BinTree(size);  copy\_tree(&tree, &neg\_tree, tree.root);  std::cout << " - Direct pass: ";  neg\_tree.print(BinTree::P\_DIRECT);  std::cout << " - Parity pass: ";  neg\_tree.print(BinTree::P\_PARITY);  std::cout << " - Reverse pass: ";  neg\_tree.print(BinTree::P\_REVERSE);    return 0;  } | #include <iostream>  #include "bin\_tree.h"  void BinTree::\_create\_node(Node\*& node, int value) {  if (node == NULL) {  node = new Node;  node->data = value;  node->left = NULL;  node->right = NULL;  }  }  void BinTree::\_print\_node(Node\*& node, int& variant) {  if (node != NULL) {  if (variant == 0) std::cout << node->data << " "; // Прямой  \_print\_node(node->left, variant);  if (variant == 1) std::cout << node->data << " "; // Симметричный  \_print\_node(node->right, variant);  if (variant == 2) std::cout << node->data << " "; // Обратный  }  }  void BinTree::\_add(Node\*& node, int& value) {  if (node == NULL) \_create\_node(node, value);  else {  switch (rand() % 2) {  case 0: {  if (node->left) \_add(node->left, value);  else \_create\_node(node->left, value);  break;  }  case 1: {  if (node->right) \_add(node->right, value);  else \_create\_node(node->right, value);  break;  }  }  }  }  BinTree::BinTree(int size) {  root = NULL;  this->size = size;  }  void BinTree::add(int value) {  \_add(this->root, value);  }  void BinTree::generate() {  for (int i = 0; i < this->size; i++) {  int value = pow(-1, rand() % 2) \* (rand() % 10);  add(value);  }  std::cout << std::endl;  }  void BinTree::print(int var) {  \_print\_node(this->root, var);  std::cout << std::endl;  } | #pragma once  #include <iostream>  class BinTree {  private:  struct Node {  int data;  Node\* left, \* right;  };  void \_create\_node(Node\*&, int);  void \_print\_node(Node\*&, int&);  void \_add(Node\*&, int&);  friend void copy\_tree(BinTree\*, BinTree\*, Node\*);  public:  int size;  Node\* root;  static const int P\_DIRECT = 0;  static const int P\_PARITY = 1;  static const int P\_REVERSE = 2;  BinTree(int);  void add(int);  void generate();  void print(int);  }; |

Алгоритм работы:

Программа создаёт новый объект дерева случайного размера и вызывает функцию генерации содержимого дерева. Функция генерации равное количеству элементов раз вызывает функцию случайного добавления в дерево значения, которая начиная с корня в случайном направлении от корня проходит дерево в поиске свободного места в нем и записывает туда значение. Затем программа три раза вызывает функцию печати значений дерева на экран: в первый раз – прямым обходом, второй – симметричным, третий – обратным.

Затем в программе создаётся новый объект дерева, в котором будут храниться отрицательные элементы. После программа вызывает функцию, которая прямым обходом обходит раннее случайным образом созданное дерево и каждое отрицательное значение вносит в только что созданное дерево, при этом также случайным образом. После значения нового дерева выводятся на экран тремя вариантами обхода.

Результат работы:



