**МИНОБРНАУКИ РОССИИ**

Федеральное государственное бюджетное

образовательное учреждение высшего образования

**«Ухтинский государственный технический университет»**

**(УГТУ)**

Кафедра вычислительной техники, информационных систем и технологий

**РАСЧЕТНО-ГРАФИЧЕСКАЯ РАБОТА №1**

Дисциплина: «Объектно-ориентированное программирование»

Шифр 191407 Группа ИСТ-2-19 Курс 2

Морданов Егор Владимирович

Проверил:

доцент кафедры ВТИСиТ Г. Н. Гатин

Ухта

2021

СОДЕРЖАНИЕ

[ЗАДАНИЕ 3](#_Toc69298732)

[ДЕМОНСТРАЦИЯ РАБОТЫ КЛАССА 6](#_Toc69298733)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 9](#_Toc69298734)

[ПРИЛОЖЕНИЕ 1 10](#_Toc69298735)

# ЗАДАНИЕ

Формулировка задания:

Создать класс «дерево».

Выполнение задания:

Для начала определим, что такое дерево. Дерево – структура данных, представляющая собой древовидную структуру в виде набора связанных узлов.

Бинарное дерево — это конечное множество элементов, которое либо пусто, либо содержит элемент (корень), связанный с двумя различными бинарными деревьями, называемыми левым и правым поддеревьями. Каждый элемент бинарного дерева называется узлом. Связи между узлами дерева называются его ветвями.

Способ представления бинарного дерева: Рисунок 1.

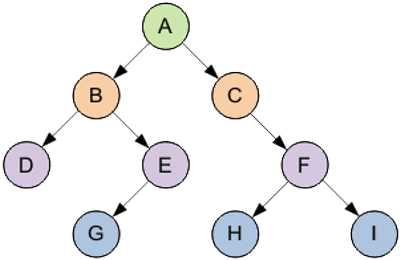


Рисунок 1 - пример бинарного дерева

Таким образом, дерево в нашей программе будет записана в виде бинарного дерева. А так как мы будем работать только с целыми числами. То тип элементов в дереве будет integer.

Дальше определим основные операции над деревом:

* Добавление элемента в дерево
* Поиск элемента в дереве
* Очищение всего дерева
* Удаление элемента в дереве
* Автоматическое создания небольшого дерева

Они будут представлены в виде методов нашего класса. Ниже представлена структура класса.

Листинг 1 - Структура класса

|  |
| --- |
| class binarytree {  private:  struct node {};  node \*root;  int level(node \*cur)  void dfs(int x, int y, node \*cur, vector<vector<int>> &ar, int step)  node \*upperBound(node \*cur, int v)  public:  binarytree(int v = 0)  void add(int v)  bool search(int v)  void deleteNode(int v)  vector<vector<int>>showtree() |

Далее приводятся описания интерфейсов основных подпрограмм:

* Основных конструктор: binarytree(int v = 0) – для инициализации дерева и создания корня в бинарном дереве;
* Оператор добавления: add(int v) – данная функция нужна для добавления элементов в порядке логики бинарного дерева;
* Метод search(int v) – возвращает true или false на то что есть элемент в дереве или нет;
* Методы deleteNode(int v) – удаляет элемент в дереве и перестраивает его в соответствии с логикой бинарного дерева;
* Метод vector<vector<int>>showtree() – функция для более красивого отображения элементов в программе, путём создания матрицы
* Метод \*upperBound(node \*cur, int v)– логика построения самого бинарного дерева;
* Метод level(node \*cur)– для определения уровня дерева
* Метод dfs(int x, int y, node \*cur, vector<vector<int>> &ar, int step) – для распределения и построения матрицы бинарного дерева.

# ДЕМОНСТРАЦИЯ РАБОТЫ КЛАССА

Демонстрация оболочки программы (Рисунок 2):

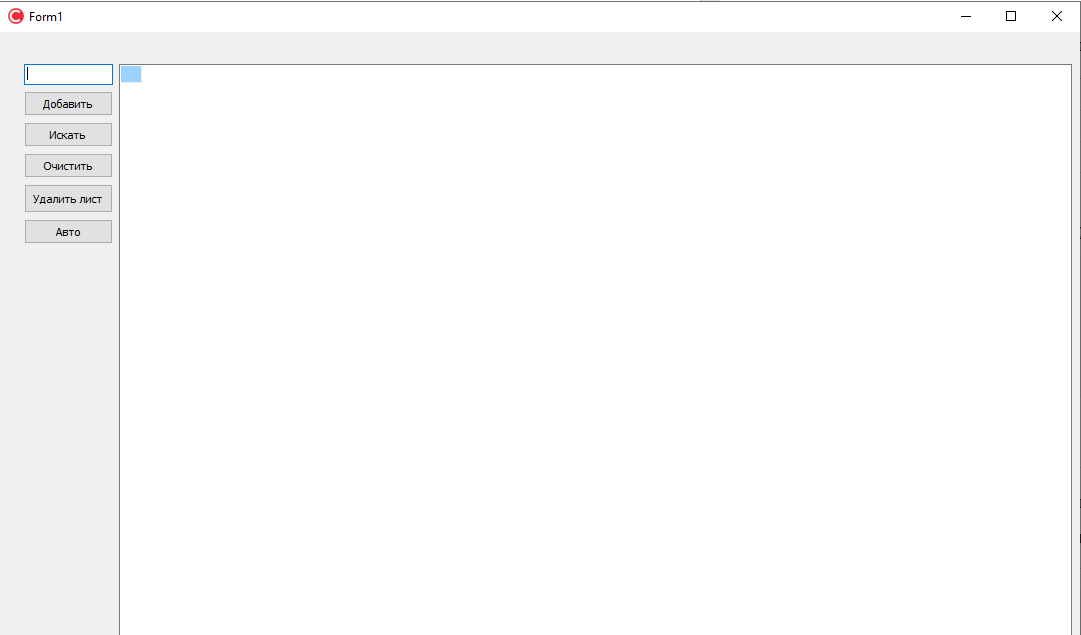


Рисунок 2 – сама программа

Функция автозаполнения дерева в программе (Рисунок 3):

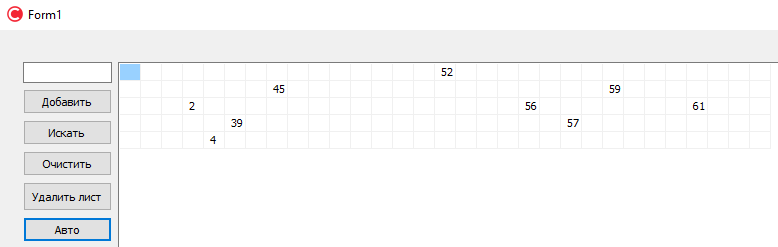


Рисунок 3 – результат автозаполнения бинарного дерева

Операция добавления элемента 55 в дерево (Рисунок 4):

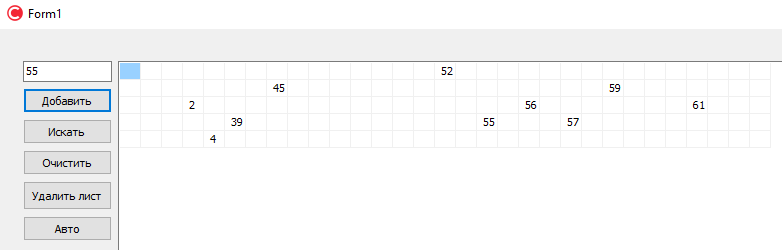


Рисунок 4 - результат добавления элемента в дерево

Операция удаления элемента 45 в дереве (Рисунок 5):

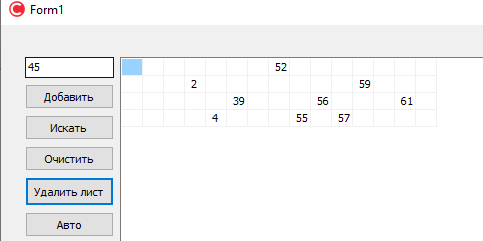


Рисунок 5 – результат удаления элемента в дереве

Поиск элемента 61 в дереве (Рисунок 6):

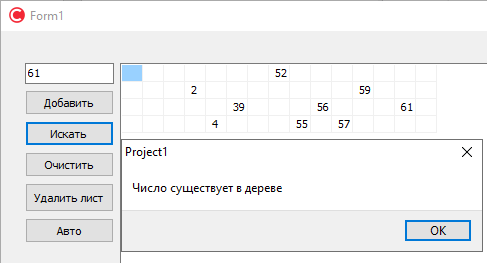


Рисунок 6 – результат поиска в дереве

Вычисление поиск элемента 62 в дереве (Рисунок 7):

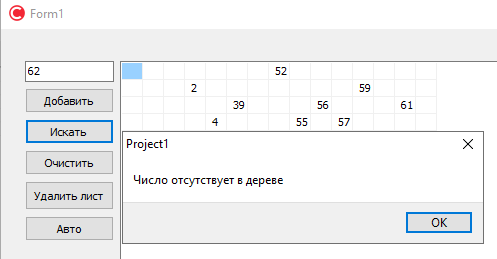


Рисунок 7 - результат поиска в дереве

Операция очистки дерева (Рисунок 8):

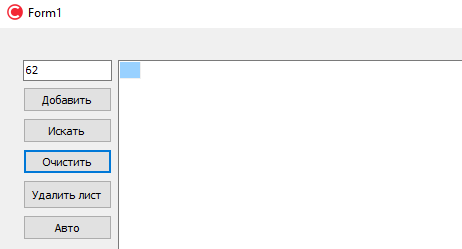


Рисунок 8 – результат очистки дерева

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Шилдт Г. Самоучитель С++: Учебное пособие / Шилдт Г. - СПб.: БХВ-Петербург, 2007. - 688 с.

ПРИЛОЖЕНИЕ 1

Листинг 2 - Заголовочный файл класса

|  |
| --- |
| #include <algorithm>  #include <vector>  #ifndef treeH  #define treeH  using namespace std;  class binarytree {  private:  struct node {  int value;  node \*prev = nullptr, \*nextL = nullptr, \*nextR = nullptr;  node(int v = 0) {  value = v;  }  };  node \*root;  int level(node \*cur) {  int mx = 1;  if (cur->nextL != nullptr)  mx = max(mx, level(cur->nextL) + 1);  if (cur->nextR != nullptr)  mx = max(mx, level(cur->nextR) + 1);  return mx;  } |

Продолжение листинга 2

|  |
| --- |
| void dfs(int x, int y, node \*cur, vector<vector<int>> &ar, int step) {  ar[x][y] = cur->value;  if (cur->nextL != nullptr)  dfs(x + 1, y - (1 << step), cur->nextL, ar, step - 1);  if (cur->nextR != nullptr)  dfs(x + 1, y + (1 << step), cur->nextR, ar, step - 1);  }  node \*upperBound(node \*cur, int v) {  node \*cnode = nullptr;  int cv = 0x3f3f3f3f;  while (true) {  if (v == cur->value)  return cur;  if (v < cur->value && cv > cur->value)  cv = cur->value, cnode = cur;  if (v <= cur->value) {  if (cur->nextL == nullptr)  return cnode;  else  cur = cur->nextL;  }  else {  if (cur->nextR == nullptr)  return cnode;  else  cur = cur->nextR;  }  }  } |

Продолжение листинга 2

|  |
| --- |
| public:  binarytree(int v = 0) {  root = new node(v);  }  void add(int v) {  node \*cur = this->root;  while (true) {  if (v == cur->value)  return;  if (v <= cur->value) {  if (cur->nextL == nullptr) {  cur->nextL = new node(v);  cur->nextL->prev = cur;  return;  }  else  cur = cur->nextL;  }  else {  if (cur->nextR == nullptr) {  cur->nextR = new node(v);  cur->nextR->prev = cur;  return;  }  else  cur = cur->nextR;  }  }  } |

Продолжение листинга 2

|  |
| --- |
| bool search(int v) {  node \*cur = upperBound(this->root, v);  return cur != nullptr && cur->value == v;  }  void deleteNode(int v) {  node\*cur = upperBound(this->root, v);  if (cur == nullptr || cur->value != v)  return;  if (cur->nextL == nullptr && cur->nextR == nullptr) {  if (cur->prev == nullptr)  exit(0);  node\* parent = cur->prev;  if (parent->nextL == cur)  parent->nextL = nullptr;  else  parent->nextR = nullptr;  delete cur;  }  else if (bool(cur->nextL != nullptr) + bool(cur->nextR != nullptr) == 1) {  node\* parent = cur->prev, \*newcur;  if (cur->nextL != nullptr)  newcur = cur->nextL;  else  newcur = cur->nextR;  if (newcur->nextL != nullptr)  newcur->nextL->prev = cur;  if (newcur->nextR != nullptr) |

Продолжение листинга 2

|  |
| --- |
| newcur->nextR->prev = cur;  cur->nextL = newcur->nextL;  cur->nextR = newcur->nextR;  cur->value = newcur->value;  delete newcur;  }  else {  node \*parent = cur->prev;  node \*mnR = upperBound(cur->nextR, cur->value);  if (mnR->nextL != nullptr)  mnR->nextL->prev = cur;  if (mnR->nextR != nullptr)  mnR->nextR->prev = cur;  node \*mnPar = mnR->prev;  if (mnPar->nextL == mnR)  mnPar->nextL = nullptr;  else  mnPar->nextR = nullptr;  if (cur->nextR == nullptr)  cur->nextR = mnR->nextR;  cur->value = mnR->value;  delete mnR;  }  }  vector<vector<int>>showtree() {  int n = level(this->root), m = (1 << n) - 1;  vector<vector<int>>ar(n, vector<int>(m)); |

Продолжение листинга 2

|  |
| --- |
| for (int i=0; i < n; i++) {  for (int j=0; j < m; j++){  ar[i][j]=0x3f3f3f3f;  }  }  dfs(0, m / 2, this->root, ar, n - 2);  return ar;  }  };  #endif |

Листинг 3 - Файл с реализацией методов класса

|  |
| --- |
| /\* Бинарное дерево поиск Binary Search Tree  Морданов Егор \*/  #include <vcl.h>  #pragma hdrstop  #include "Unit1.h"  #include "tree.h"  #include <random>  #include <ctime>  //---------------------------------------------------------------------------  #pragma package(smart\_init)  #pragma resource "\*.dfm"  using namespace std;  TForm1 \*Form1;  int numconvert(string s) {  int ans = 0;  for (int i = 0; i < s.size(); ++i) { |

Продолжение листинга 3

|  |
| --- |
| if (s[i] >= '0' && s[i] <= '9') {  ans \*= 10;  ans += s[i] - '0';  }  else  return 0;  }  return ans;  }  binarytree \*tree = nullptr;  void showtrees();  //---------------------------------------------------------------------------  \_\_fastcall TForm1::TForm1(TComponent\* Owner)  : TForm(Owner)  {  }  //---------------------------------------------------------------------------  void \_\_fastcall TForm1::Button1Click(TObject \*Sender)  {  if (tree == nullptr)  tree= new binarytree(numconvert(AnsiString(Edit1 -> Text).c\_str()));  else  tree->add(numconvert(AnsiString(Edit1 -> Text).c\_str()));  showtrees();  }  //---------------------------------------------------------------------------  void \_\_fastcall TForm1::Button2Click(TObject \*Sender) |

Продолжение листинга 3

|  |
| --- |
| {  bool check = tree->search(numconvert(AnsiString(Edit1 -> Text).c\_str()));  if (check)  ShowMessage("Число существует в дереве");  else  ShowMessage("Число отсутствует в дереве");  }  //---------------------------------------------------------------------------  void showtrees(){  vector<vector<int>>ar = tree->showtree();  Form1->StringGrid1 -> ColCount = ar[0].size();  Form1->StringGrid1 -> RowCount = ar.size();  for (int i=0; i < ar[0].size(); i++) {  Form1->StringGrid1 -> Rows[i]->Clear();  }  Form1->StringGrid1 -> ColCount = ar[0].size();  Form1->StringGrid1 -> RowCount = ar.size();  for (int i=0; i < ar[0].size(); i++) {  Form1->StringGrid1 -> Rows[i]->Clear();  }  for (int i=0; i < min(int(ar.size()), 999); i++) { // при очень большом дереве  for (int j=0; j < min(int(ar[0].size()), 999); j++){ //stringgrid не выдерживает  if (ar[i][j]!=0x3f3f3f3f) //за то красивое отображение |

Продолжение листинга 3

|  |
| --- |
| Form1->StringGrid1 -> ColCount = ar[0].size();  Form1->StringGrid1 -> RowCount = ar.size();  for (int i=0; i < ar[0].size(); i++) {  Form1->StringGrid1 -> Rows[i]->Clear();  }  for (int i=0; i < min(int(ar.size()), 999); i++) { // при очень большом дереве  for (int j=0; j < min(int(ar[0].size()), 999); j++){ //stringgrid не выдерживает  if (ar[i][j]!=0x3f3f3f3f) //за то красивое отображение  Form1->StringGrid1 -> Cells[j][i] = ar[i][j];  }  }  }  //---------------------------------------------------------------------------  void \_\_fastcall TForm1::Button4Click(TObject \*Sender)  {  delete tree;  Form1->StringGrid1 -> ColCount = 0;  Form1->StringGrid1 -> RowCount = 0;  }  //---------------------------------------------------------------------------  void \_\_fastcall TForm1::Button3Click(TObject \*Sender)  {  mt19937 r(time(0));  delete tree;  tree = new binarytree(r() % 50+25) |

Продолжение листинга 3

|  |
| --- |
| for (int i = 0; i < 8; ++i)  tree->add(r() % 100);  showtrees();  }  //---------------------------------------------------------------------------  void \_\_fastcall TForm1::Button5Click(TObject \*Sender)  {  tree->deleteNode(numconvert(AnsiString(Edit1 -> Text).c\_str()));  showtrees();  }  //--------------------------------------------------------------------------- |