Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Лабораторная работа 11

По дисциплине «Основы алгоритмизации программирования»

На тему «Бинарные деревья»

Выполнил:

Студент 1 курса 6 группы

Кучерук Николай Петрович

Преподаватель: асс. Андронова М.В.

2024, Минск

**Задание**

| **№ варианта** | **Условие задачи** |
| --- | --- |
| **9** | Дан указатель **p1** на корень непустого дерева и число **k**. Написать функцию вывода количества вершин дерева, значение которых равно **k**. |

|  |
| --- |
| **Код программы** |
| #include <ctime>  #include <iostream>  using namespace std;  struct Tree //дерево  {  int key; //ключ  int data; //информация для хранения  Tree\* Left, \* Right;  };  Tree\* makeTree(Tree\* Root); //Создание дерева  Tree\* list(int i, int); //Создание нового элемента  Tree\* insertElem(Tree\* Root, int key, int a); //Добавление нового элемента  Tree\* search(Tree\* n, int key); //Поиск элемента по ключу  Tree\* delet(Tree\* Root, int key); //Удаление элемента по ключу  int view(Tree\* t, int level); //Вывод дерева  int count\_same(Tree\* t, int); //Подсчет одинаковых значений  bool delAll(Tree\* t); //Очистка дерева  int c = 0; //количество слов  Tree\* Root = NULL; //указатель на корень  void main()  {  setlocale(0, "Russian");  int key, choice, n,data;  Tree\* rc;  for (;;)  {  cout << "1 - создание дерева\n";  cout << "2 - добавление элемента\n";  cout << "3 - поиск по ключу\n";  cout << "4 - удаление элемента\n";  cout << "5 - вывод дерева\n";  cout << "6 - подсчет количества совпадающих цифр\n";  cout << "7 - очистка дерева\n";  cout << "8 - выход\n";  cout << "ваш выбор?\n";  cin >> choice;  cout << "\n";  switch (choice)  {  case 1: Root = makeTree(Root); break;  case 2: cout << "\nВведите ключ: ";  cin >> key;  data = rand() % 7;  insertElem(Root, key, data); break;  case 3: cout << "\nВведите ключ: ";  cin >> key;  rc = search(Root, key);  cout << "Найденное слово= ";  cout<<rc->data; break;  case 4: cout << "\nВведите удаляемый ключ: ";  cin >> key;  Root = delet(Root, key); break;  case 5: if (Root->key >= 0)  {  cout << "Дерево повернуто на 90 град. влево" << endl;  view(Root, 0);  }  else cout << "Дерево пустое\n"; break;  case 6: cout << "\nВведите цифру: ";  cin >> n;  cout << "Кол-во подходящих значений" << count\_same(Root, n) << endl << endl;  view(Root, 0);  break;  case 7: delAll(Root); break;  case 8: exit(0);  }  }  }  Tree\* makeTree(Tree\* Root) //Создание дерева  {  int key,data;  cout << "Конец ввода - отрицательное число\n\n";  if (Root == NULL) // если дерево не создано  {  cout << "Введите ключ корня: ";  cin >> key;  data = rand() % 7;  Root = list(key, data); // установка указателя на корень  }  while (1) //добавление элементов  {  cout << "\nВведите ключ: ";  cin >> key;  if (key < 0) break; //признак выхода (ключ < 0)  data = rand() % 7;  insertElem(Root, key, data);  }  return Root;  }  Tree\* list(int i, int data) //Создание нового элемента  {  Tree\* t = new Tree[sizeof(Tree)];  t->key = i;  t->data = data;  t->Left = t->Right = NULL;  return t;  }  Tree\* insertElem(Tree\* t, int key, int data) //Добавление нового элемента  {  Tree\* Prev=t; // Prev - элемент перед текущим  int find = 0; // признак поиска  while (t && !find)  {  Prev = t;  if (key == t->key)  find = 1; //ключи должны быть уникальны  else  if (key < t->key)  t = t->Left;  else  t = t->Right;  }  if (!find) //найдено место с адресом Prev  {  t = list(key, data); //создается новый узел  if (key < Prev->key) // и присоединяется либо  Prev->Left = t; //переход на левую ветвь,  else  Prev->Right = t; // либо на правую  }  return t;  }  Tree\* delet(Tree\* Root, int key) //Удаление элемента по ключу  { // Del, Prev\_Del - удаляемый элемент и его предыдущий;  // R, Prev\_R - элемент, на который заменяется удаленный, и его родитель;  Tree\* Del, \* Prev\_Del, \* R, \* Prev\_R;  Del = Root;  Prev\_Del = NULL;  while (Del != NULL && Del->key != key)//поиск элемента и его родителя  {  Prev\_Del = Del;  if (Del->key > key)  Del = Del->Left;  else  Del = Del->Right;  }  if (Del == NULL) // элемент не найден  {  puts("\nНет такого ключа");  return Root;  }  if (Del->Right == NULL) // поиск элемента R для замены  R = Del->Left;  else  if (Del->Left == NULL)  R = Del->Right;  else  {  Prev\_R = Del; //поиск самого правого элемента в левом поддереве  R = Del->Left;  while (R->Right != NULL)  {  Prev\_R = R;  R = R->Right;  }  if (Prev\_R == Del) // найден элемент для замены R и его родителя Prev\_R  R->Right = Del->Right;  else  {  R->Right = Del->Right;  Prev\_R->Right = R->Left;  R->Left = Prev\_R;  }  }  if (Del == Root) Root = R; //удаление корня и замена его на R  else  // поддерево R присоединяется к родителю удаляемого узла  if (Del->key < Prev\_Del->key)  Prev\_Del->Left = R; //на левую ветвь  else  Prev\_Del->Right = R; //на правую ветвь  int tmp = Del->key;  cout << "\nУдален элемент с ключом " << tmp << endl;  delete Del;  return Root;  }  Tree\* search(Tree\* n, int key) //Поиск элемента по ключу  {  Tree\* rc = n;  if (rc != NULL)  {  if (key < (key, n->key))  rc = search(n->Left, key);  else  if (key > (key, n->key))  rc = search(n->Right, key);  }  else  cout << "Нет такого элемента\n";  return rc;  }  int count\_same(Tree\* t, int k) {//Подсчет одинаковых значений  if (t == NULL) {  return 0;  }  int count = 0;  if (t->data == k) {  count = 1;  }  return count + count\_same(t->Left, k) + count\_same(t->Right, k);  }  int view(Tree\* t, int level) //Вывод дерева  {  if (t)  {  view(t->Right, level + 1); //вывод правого поддерева  for (int i = 0; i < level; i++)  cout << " ";  int tm = t->key;  cout << tm << ' '<<t->data<<endl;  view(t->Left, level + 1); //вывод левого поддерева  return 0;  }  return 1;  }  bool delAll(Tree\* t) //Очистка дерева  {  if (t != NULL)  {  delAll(t->Left);  delAll(t->Right);  delete t;  return true;  }  return false;  } |
| **Результат программы** |
|  |

Дополнительные задания

|  |  |
| --- | --- |
| **7** | Дан указатель **p1** на корень непустого дерева. Написать функцию вывода количества листьев данного дерева. |

|  |
| --- |
| **Код программы** |
| #include <ctime>  #include <iostream>  using namespace std;  struct Tree //дерево  {  int key; //ключ  int data; //информация для хранения  Tree\* Left, \* Right;  };  Tree\* makeTree(Tree\* Root); //Создание дерева  Tree\* list(int i, int); //Создание нового элемента  Tree\* insertElem(Tree\* Root, int key, int a); //Добавление нового элемента  Tree\* search(Tree\* n, int key); //Поиск элемента по ключу  Tree\* delet(Tree\* Root, int key); //Удаление элемента по ключу  int view(Tree\* t, int level); //Вывод дерева  int count\_leaves(Tree\* t); //Подсчет крайних звеньев  bool delAll(Tree\* t); //Очистка дерева  int c = 0; //количество слов  Tree\* Root = NULL; //указатель на корень  void main()  {  setlocale(0, "Russian");  int key, choice, n,data;  Tree\* rc;  for (;;)  {  cout << "1 - создание дерева\n";  cout << "2 - добавление элемента\n";  cout << "3 - поиск по ключу\n";  cout << "4 - удаление элемента\n";  cout << "5 - вывод дерева\n";  cout << "6 - подсчет ''листьев''\n";  cout << "7 - очистка дерева\n";  cout << "8 - выход\n";  cout << "ваш выбор?\n";  cin >> choice;  cout << "\n";  switch (choice)  {  case 1: Root = makeTree(Root); break;  case 2: cout << "\nВведите ключ: ";  cin >> key;  data = rand() % 7;  insertElem(Root, key, data); break;  case 3: cout << "\nВведите ключ: ";  cin >> key;  rc = search(Root, key);  cout << "Найденное слово= ";  cout<<rc->data; break;  case 4: cout << "\nВведите удаляемый ключ: ";  cin >> key;  Root = delet(Root, key); break;  case 5: if (Root->key >= 0)  {  cout << "Дерево повернуто на 90 град. влево" << endl;  view(Root, 0);  }  else cout << "Дерево пустое\n"; break;  case 6:  cout << "Кол-во листьев:" << count\_leaves(Root) << endl << endl;  break;  case 7: delAll(Root); break;  case 8: exit(0);  }  }  }  Tree\* makeTree(Tree\* Root) //Создание дерева  {  int key,data;  cout << "Конец ввода - отрицательное число\n\n";  if (Root == NULL) // если дерево не создано  {  cout << "Введите ключ корня: ";  cin >> key;  data = rand() % 7;  Root = list(key, data); // установка указателя на корень  }  while (1) //добавление элементов  {  cout << "\nВведите ключ: ";  cin >> key;  if (key < 0) break; //признак выхода (ключ < 0)  data = rand() % 7;  insertElem(Root, key, data);  }  return Root;  }  Tree\* list(int i, int data) //Создание нового элемента  {  Tree\* t = new Tree[sizeof(Tree)];  t->key = i;  t->data = data;  t->Left = t->Right = NULL;  return t;  }  Tree\* insertElem(Tree\* t, int key, int data) //Добавление нового элемента  {  Tree\* Prev=t; // Prev - элемент перед текущим  int find = 0; // признак поиска  while (t && !find)  {  Prev = t;  if (key == t->key)  find = 1; //ключи должны быть уникальны  else  if (key < t->key)  t = t->Left;  else  t = t->Right;  }  if (!find) //найдено место с адресом Prev  {  t = list(key, data); //создается новый узел  if (key < Prev->key) // и присоединяется либо  Prev->Left = t; //переход на левую ветвь,  else  Prev->Right = t; // либо на правую  }  return t;  }  Tree\* delet(Tree\* Root, int key) //Удаление элемента по ключу  { // Del, Prev\_Del - удаляемый элемент и его предыдущий;  // R, Prev\_R - элемент, на который заменяется удаленный, и его родитель;  Tree\* Del, \* Prev\_Del, \* R, \* Prev\_R;  Del = Root;  Prev\_Del = NULL;  while (Del != NULL && Del->key != key)//поиск элемента и его родителя  {  Prev\_Del = Del;  if (Del->key > key)  Del = Del->Left;  else  Del = Del->Right;  }  if (Del == NULL) // элемент не найден  {  puts("\nНет такого ключа");  return Root;  }  if (Del->Right == NULL) // поиск элемента R для замены  R = Del->Left;  else  if (Del->Left == NULL)  R = Del->Right;  else  {  Prev\_R = Del; //поиск самого правого элемента в левом поддереве  R = Del->Left;  while (R->Right != NULL)  {  Prev\_R = R;  R = R->Right;  }  if (Prev\_R == Del) // найден элемент для замены R и его родителя Prev\_R  R->Right = Del->Right;  else  {  R->Right = Del->Right;  Prev\_R->Right = R->Left;  R->Left = Prev\_R;  }  }  if (Del == Root) Root = R; //удаление корня и замена его на R  else  // поддерево R присоединяется к родителю удаляемого узла  if (Del->key < Prev\_Del->key)  Prev\_Del->Left = R; //на левую ветвь  else  Prev\_Del->Right = R; //на правую ветвь  int tmp = Del->key;  cout << "\nУдален элемент с ключом " << tmp << endl;  delete Del;  return Root;  }  Tree\* search(Tree\* n, int key) //Поиск элемента по ключу  {  Tree\* rc = n;  if (rc != NULL)  {  if (key < (key, n->key))  rc = search(n->Left, key);  else  if (key > (key, n->key))  rc = search(n->Right, key);  }  else  cout << "Нет такого элемента\n";  return rc;  }  int count\_leaves(Tree\* t) {//подсчет крайних звеньев  int count = 0;  if (t == nullptr) {  return 0;  }  if (t->Left == NULL && t->Right == NULL) {  return 1;  }  count += count\_leaves(t->Left);  count += count\_leaves(t->Right);  return count;  }  int view(Tree\* t, int level) //Вывод дерева  {  if (t)  {  view(t->Right, level + 1); //вывод правого поддерева  for (int i = 0; i < level; i++)  cout << " ";  int tm = t->key;  cout << tm << ' '<<t->data<<endl;  view(t->Left, level + 1); //вывод левого поддерева  return 0;  }  return 1;  }  bool delAll(Tree\* t) //Очистка дерева  {  if (t != NULL)  {  delAll(t->Left);  delAll(t->Right);  delete t;  return true;  }  return false;  } |
| **Результат программы** |
|  |

|  |  |
| --- | --- |
| **10** | Дан указатель **p1** на корень непустого дерева. Написать функцию вывода вывода суммы значений всех вершин данного дерева. |

|  |
| --- |
| **Код программы** |
| #include <ctime>  #include <iostream>  using namespace std;  struct Tree //дерево  {  int key; //ключ  int data; //информация для хранения  Tree\* Left, \* Right;  };  Tree\* makeTree(Tree\* Root); //Создание дерева  Tree\* list(int i, int); //Создание нового элемента  Tree\* insertElem(Tree\* Root, int key, int a); //Добавление нового элемента  Tree\* search(Tree\* n, int key); //Поиск элемента по ключу  Tree\* delet(Tree\* Root, int key); //Удаление элемента по ключу  int view(Tree\* t, int level); //Вывод дерева  int count\_sum(Tree\* t); //Подсчет суммы  bool delAll(Tree\* t); //Очистка дерева  int c = 0; //количество слов  Tree\* Root = NULL; //указатель на корень  void main()  {  setlocale(0, "Russian");  int key, choice, n,data;  Tree\* rc;  for (;;)  {  cout << "1 - создание дерева\n";  cout << "2 - добавление элемента\n";  cout << "3 - поиск по ключу\n";  cout << "4 - удаление элемента\n";  cout << "5 - вывод дерева\n";  cout << "6 - подсчет суммы элементов\n";  cout << "7 - очистка дерева\n";  cout << "8 - выход\n";  cout << "ваш выбор?\n";  cin >> choice;  cout << "\n";  switch (choice)  {  case 1: Root = makeTree(Root); break;  case 2: cout << "\nВведите ключ: ";  cin >> key;  data = rand() % 7;  insertElem(Root, key, data); break;  case 3: cout << "\nВведите ключ: ";  cin >> key;  rc = search(Root, key);  cout << "Найденное слово= ";  cout<<rc->data; break;  case 4: cout << "\nВведите удаляемый ключ: ";  cin >> key;  Root = delet(Root, key); break;  case 5: if (Root->key >= 0)  {  cout << "Дерево повернуто на 90 град. влево" << endl;  view(Root, 0);  }  else cout << "Дерево пустое\n"; break;  case 6:  cout << "Сумма элементов:" << count\_sum(Root) << endl << endl;  break;  case 7: delAll(Root); break;  case 8: exit(0);  }  }  }  Tree\* makeTree(Tree\* Root) //Создание дерева  {  int key,data;  cout << "Конец ввода - отрицательное число\n\n";  if (Root == NULL) // если дерево не создано  {  cout << "Введите ключ корня: ";  cin >> key;  data = rand() % 7;  Root = list(key, data); // установка указателя на корень  }  while (1) //добавление элементов  {  cout << "\nВведите ключ: ";  cin >> key;  if (key < 0) break; //признак выхода (ключ < 0)  data = rand() % 7;  insertElem(Root, key, data);  }  return Root;  }  Tree\* list(int i, int data) //Создание нового элемента  {  Tree\* t = new Tree[sizeof(Tree)];  t->key = i;  t->data = data;  t->Left = t->Right = NULL;  return t;  }  Tree\* insertElem(Tree\* t, int key, int data) //Добавление нового элемента  {  Tree\* Prev=t; // Prev - элемент перед текущим  int find = 0; // признак поиска  while (t && !find)  {  Prev = t;  if (key == t->key)  find = 1; //ключи должны быть уникальны  else  if (key < t->key)  t = t->Left;  else  t = t->Right;  }  if (!find) //найдено место с адресом Prev  {  t = list(key, data); //создается новый узел  if (key < Prev->key) // и присоединяется либо  Prev->Left = t; //переход на левую ветвь,  else  Prev->Right = t; // либо на правую  }  return t;  }  Tree\* delet(Tree\* Root, int key) //Удаление элемента по ключу  { // Del, Prev\_Del - удаляемый элемент и его предыдущий;  // R, Prev\_R - элемент, на который заменяется удаленный, и его родитель;  Tree\* Del, \* Prev\_Del, \* R, \* Prev\_R;  Del = Root;  Prev\_Del = NULL;  while (Del != NULL && Del->key != key)//поиск элемента и его родителя  {  Prev\_Del = Del;  if (Del->key > key)  Del = Del->Left;  else  Del = Del->Right;  }  if (Del == NULL) // элемент не найден  {  puts("\nНет такого ключа");  return Root;  }  if (Del->Right == NULL) // поиск элемента R для замены  R = Del->Left;  else  if (Del->Left == NULL)  R = Del->Right;  else  {  Prev\_R = Del; //поиск самого правого элемента в левом поддереве  R = Del->Left;  while (R->Right != NULL)  {  Prev\_R = R;  R = R->Right;  }  if (Prev\_R == Del) // найден элемент для замены R и его родителя Prev\_R  R->Right = Del->Right;  else  {  R->Right = Del->Right;  Prev\_R->Right = R->Left;  R->Left = Prev\_R;  }  }  if (Del == Root) Root = R; //удаление корня и замена его на R  else  // поддерево R присоединяется к родителю удаляемого узла  if (Del->key < Prev\_Del->key)  Prev\_Del->Left = R; //на левую ветвь  else  Prev\_Del->Right = R; //на правую ветвь  int tmp = Del->key;  cout << "\nУдален элемент с ключом " << tmp << endl;  delete Del;  return Root;  }  Tree\* search(Tree\* n, int key) //Поиск элемента по ключу  {  Tree\* rc = n;  if (rc != NULL)  {  if (key < (key, n->key))  rc = search(n->Left, key);  else  if (key > (key, n->key))  rc = search(n->Right, key);  }  else  cout << "Нет такого элемента\n";  return rc;  }  int count\_sum(Tree\* t) {//подсчет суммы  int count = 0;  if (t == nullptr) {  return 0;  }    count += count\_sum(t->Left);  count += count\_sum(t->Right);  return count+t->data;  }  int view(Tree\* t, int level) //Вывод дерева  {  if (t)  {  view(t->Right, level + 1); //вывод правого поддерева  for (int i = 0; i < level; i++)  cout << " ";  int tm = t->key;  cout << tm << ' '<<t->data<<endl;  view(t->Left, level + 1); //вывод левого поддерева  return 0;  }  return 1;  }  bool delAll(Tree\* t) //Очистка дерева  {  if (t != NULL)  {  delAll(t->Left);  delAll(t->Right);  delete t;  return true;  }  return false;  } |
| **Результат программы** |
|  |

|  |  |
| --- | --- |
| **16** | Дан указатель **p1** на корень непустого дерева. Написать функцию вывода суммы значений всех листьев данного дерева. |

|  |
| --- |
| **Код программы** |
| #include <ctime>  #include <iostream>  using namespace std;  struct Tree //дерево  {  int key; //ключ  int data; //информация для хранения  Tree\* Left, \* Right;  };  Tree\* makeTree(Tree\* Root); //Создание дерева  Tree\* list(int i, int); //Создание нового элемента  Tree\* insertElem(Tree\* Root, int key, int a); //Добавление нового элемента  Tree\* search(Tree\* n, int key); //Поиск элемента по ключу  Tree\* delet(Tree\* Root, int key); //Удаление элемента по ключу  int view(Tree\* t, int level); //Вывод дерева  int count\_leaves\_sum(Tree\* t); //Подсчет суммы крайних звеньев  bool delAll(Tree\* t); //Очистка дерева  int c = 0; //количество слов  Tree\* Root = NULL; //указатель на корень  void main()  {  setlocale(0, "Russian");  int key, choice, n,data;  Tree\* rc;  for (;;)  {  cout << "1 - создание дерева\n";  cout << "2 - добавление элемента\n";  cout << "3 - поиск по ключу\n";  cout << "4 - удаление элемента\n";  cout << "5 - вывод дерева\n";  cout << "6 - подсчет суммы крайних звеньев\n";  cout << "7 - очистка дерева\n";  cout << "8 - выход\n";  cout << "ваш выбор?\n";  cin >> choice;  cout << "\n";  switch (choice)  {  case 1: Root = makeTree(Root); break;  case 2: cout << "\nВведите ключ: ";  cin >> key;  data = rand() % 7;  insertElem(Root, key, data); break;  case 3: cout << "\nВведите ключ: ";  cin >> key;  rc = search(Root, key);  cout << "Найденное слово= ";  cout<<rc->data; break;  case 4: cout << "\nВведите удаляемый ключ: ";  cin >> key;  Root = delet(Root, key); break;  case 5: if (Root->key >= 0)  {  cout << "Дерево повернуто на 90 град. влево" << endl;  view(Root, 0);  }  else cout << "Дерево пустое\n"; break;  case 6:  cout << "Сумма крайних звеньев:" << count\_leaves\_sum(Root) << endl << endl;  break;  case 7: delAll(Root); break;  case 8: exit(0);  }  }  }  Tree\* makeTree(Tree\* Root) //Создание дерева  {  int key,data;  cout << "Конец ввода - отрицательное число\n\n";  if (Root == NULL) // если дерево не создано  {  cout << "Введите ключ корня: ";  cin >> key;  data = rand() % 7;  Root = list(key, data); // установка указателя на корень  }  while (1) //добавление элементов  {  cout << "\nВведите ключ: ";  cin >> key;  if (key < 0) break; //признак выхода (ключ < 0)  data = rand() % 7;  insertElem(Root, key, data);  }  return Root;  }  Tree\* list(int i, int data) //Создание нового элемента  {  Tree\* t = new Tree[sizeof(Tree)];  t->key = i;  t->data = data;  t->Left = t->Right = NULL;  return t;  }  Tree\* insertElem(Tree\* t, int key, int data) //Добавление нового элемента  {  Tree\* Prev=t; // Prev - элемент перед текущим  int find = 0; // признак поиска  while (t && !find)  {  Prev = t;  if (key == t->key)  find = 1; //ключи должны быть уникальны  else  if (key < t->key)  t = t->Left;  else  t = t->Right;  }  if (!find) //найдено место с адресом Prev  {  t = list(key, data); //создается новый узел  if (key < Prev->key) // и присоединяется либо  Prev->Left = t; //переход на левую ветвь,  else  Prev->Right = t; // либо на правую  }  return t;  }  Tree\* delet(Tree\* Root, int key) //Удаление элемента по ключу  { // Del, Prev\_Del - удаляемый элемент и его предыдущий;  // R, Prev\_R - элемент, на который заменяется удаленный, и его родитель;  Tree\* Del, \* Prev\_Del, \* R, \* Prev\_R;  Del = Root;  Prev\_Del = NULL;  while (Del != NULL && Del->key != key)//поиск элемента и его родителя  {  Prev\_Del = Del;  if (Del->key > key)  Del = Del->Left;  else  Del = Del->Right;  }  if (Del == NULL) // элемент не найден  {  puts("\nНет такого ключа");  return Root;  }  if (Del->Right == NULL) // поиск элемента R для замены  R = Del->Left;  else  if (Del->Left == NULL)  R = Del->Right;  else  {  Prev\_R = Del; //поиск самого правого элемента в левом поддереве  R = Del->Left;  while (R->Right != NULL)  {  Prev\_R = R;  R = R->Right;  }  if (Prev\_R == Del) // найден элемент для замены R и его родителя Prev\_R  R->Right = Del->Right;  else  {  R->Right = Del->Right;  Prev\_R->Right = R->Left;  R->Left = Prev\_R;  }  }  if (Del == Root) Root = R; //удаление корня и замена его на R  else  // поддерево R присоединяется к родителю удаляемого узла  if (Del->key < Prev\_Del->key)  Prev\_Del->Left = R; //на левую ветвь  else  Prev\_Del->Right = R; //на правую ветвь  int tmp = Del->key;  cout << "\nУдален элемент с ключом " << tmp << endl;  delete Del;  return Root;  }  Tree\* search(Tree\* n, int key) //Поиск элемента по ключу  {  Tree\* rc = n;  if (rc != NULL)  {  if (key < (key, n->key))  rc = search(n->Left, key);  else  if (key > (key, n->key))  rc = search(n->Right, key);  }  else  cout << "Нет такого элемента\n";  return rc;  }  int count\_leaves\_sum(Tree\* t) {//подсчет суммы крайних звеньев  int count = 0;  if (t == nullptr) {  return 0;  }    if (t->Left == NULL && t->Right == NULL) {  count += t->data;    }  count += count\_leaves\_sum(t->Left);  count += count\_leaves\_sum(t->Right);  return count;  }  int view(Tree\* t, int level) //Вывод дерева  {  if (t)  {  view(t->Right, level + 1); //вывод правого поддерева  for (int i = 0; i < level; i++)  cout << " ";  int tm = t->key;  cout << tm << ' '<<t->data<<endl;  view(t->Left, level + 1); //вывод левого поддерева  return 0;  }  return 1;  }  bool delAll(Tree\* t) //Очистка дерева  {  if (t != NULL)  {  delAll(t->Left);  delAll(t->Right);  delete t;  return true;  }  return false;  } |
| **Результат программы** |
|  |