Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

Лабораторная работа 11

По дисциплине «Основы алгоритмизации и программирования»

На тему «Бинарные деревья»

Выполнила:

Студентка 1 курса 2 группы

Глухова Д.В.

Преподаватель: асс. Андронова М.В.

2023, Минск

Вариант 2

3. Разработать программу работы с бинарным деревом, в которую включить основные функции манипуляции данными и функцию в соответствии со своим вариантом из таблицы, представленной ниже.

Дан указатель **p1** на корень бинарного дерева. Написать функцию вывода количества листьев дерева, которые являются правыми дочерними вершинами.

|  |
| --- |
| Исходный код |
| // Дан указатель p1 на корень бинарного дерева. Написать функцию вывода количества листьев дерева, которые являются правыми дочерними вершинами.  #include <iostream>  using namespace std;  struct Tree //дерево  {  int key; //ключ  char text[5]; //текст - не более 4 букв  Tree\* Left, \* Right; // укащатели на левую и правую ветви  };  int sum = 0, result3 = 0, counter2 = 0;  Tree\* makeTree(Tree\* Root); //Создание дерева  Tree\* list(int i, char\* s); //Создание нового элемента  Tree\* insertElem(Tree\* Root, int key, char\* s); //Добавление нового элемента  Tree\* search(Tree\* n, int key); //Поиск элемента по ключу  Tree\* delet(Tree\* Root, int key); //Удаление элемента по ключу  void view(Tree\* t, int level); //Вывод дерева  int count(Tree\* t, char letter); //Подсчет количества слов  void delAll(Tree\* t); //Очистка дерева  int check\_left(Tree\* tree); // Подсчёт правых  int c = 0; //количество слов  Tree\* Root = NULL; //указатель корня  void main()  {  setlocale(0, "Russian");  int key, choice, n, numLeft = 0;  Tree\* rc; char s[5], letter;  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 букв\n";  cout << "7 - очистка дерева\n";  cout << "8 - количество правых дочерних вершин\n";  cout << "9 - 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;  cout << "Введите слово: "; cin >> s;  insertElem(Root, key, s); break;  case 3: cout << "\nВведите ключ: "; cin >> key;  rc = search(Root, key);  cout << "Найденное слово= ";  puts(rc->text); 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 букву: "; cin >> letter;  n = count(Root, letter);  cout << "Количество слов, начинающихся с буквы " << letter;  cout << " равно " << n << endl; break;  case 7: delAll(Root); break;  case 8: numLeft = check\_left(Root);  cout << "Количество правых дочерних вершин = " << numLeft << endl;  break;  case 9: exit(0);  }  }  }  Tree\* makeTree(Tree\* Root) //Создание дерева  {  int key; char s[20]; // для ключа и строки  cout << "Конец ввода - отрицательное число\n\n";  if (Root == NULL) // если дерево не создано  {  cout << "Введите ключ корня: "; cin >> key;  cout << "Введите слово корня: "; cin >> s;  Root = list(key, s); // установка указателя на корень  }  while (1) //добавление элементов  {  cout << "\nВведите ключ: "; cin >> key;  if (key < 0) break; //признак выхода (ключ < 0)  cout << "Введите слово: "; cin >> s;  insertElem(Root, key, s);  }  return Root;  }  Tree\* list(int i, char\* s) //Создание нового элемента  {  Tree\* t = new Tree[sizeof(Tree)]; // выделение памяти  t->key = i; // ключ  for (i = 0; i < 5; i++)  \*((t->text) + i) = \*(s + i); // создание  t->Left = t->Right = NULL; // правая и левая ветви  return t;  }  Tree\* insertElem(Tree\* t, int key, char\* s) //Добавление нового элемента  {  Tree\* Prev = NULL; // 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, s); //создается новый узел  if (key < Prev->key) // если ключ меньше  Prev->Left = t; // то помещается на левую ветвь  else  Prev->Right = t; // иначе 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; // переход на левую ветвь  else  if (Del->Left == NULL)  R = Del->Right; // или на правую  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(Tree\* t, char letter) //Подсчет количества букв  {  if (t)  {  count(t->Right, letter);  if (\*(t->text) == letter)  c++; // увеличиваем счетчик  count(t->Left, letter);  }  return c;  }  void 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 << ' ';  puts(t->text);  view(t->Left, level + 1); //вывод левого поддерева  }  }  void delAll(Tree\* t) //Очистка дерева  {  if (t != NULL)  {  delAll(t->Left);  delAll(t->Right);  delete t;  }  }  int check\_left(Tree\* tree)  {  if (tree->Right != NULL)  {  check\_left(tree->Right);  sum++; // подсчет количества листьев, которые являются правыми дочерними вершинами  }  if (tree->Left != NULL)  check\_left(tree->Left);  return sum;  } |
| Консоль отладки |
|  |

Дополнительные задания

Вариант 5

Дан указатель **p1** на корень непустого дерева. Написать функцию вывода количества вершин дерева, являющихся левыми дочерними вершинами (корень дерева не учитывать).

|  |
| --- |
| Исходный код |
| // Дан указатель p1 на корень бинарного дерева. Написать функцию вывода количества листьев дерева, которые являются левыми дочерними вершинами.  #include <iostream>  using namespace std;  struct Tree //дерево  {  int key; //ключ  char text[5]; //текст - не более 4 букв  Tree\* Left, \* Right; // укащатели на левую и правую ветви  };  int sum = 0, result3 = 0, counter2 = 0;  Tree\* makeTree(Tree\* Root); //Создание дерева  Tree\* list(int i, char\* s); //Создание нового элемента  Tree\* insertElem(Tree\* Root, int key, char\* s); //Добавление нового элемента  Tree\* search(Tree\* n, int key); //Поиск элемента по ключу  Tree\* delet(Tree\* Root, int key); //Удаление элемента по ключу  void view(Tree\* t, int level); //Вывод дерева  int count(Tree\* t, char letter); //Подсчет количества слов  void delAll(Tree\* t); //Очистка дерева  int task(Tree\* tree); // Подсчёт левых  int c = 0; //количество слов  Tree\* Root = NULL; //указатель корня  void main()  {  setlocale(0, "Russian");  int key, choice, n, numLeft = 0;  Tree\* rc; char s[5], letter;  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 букв\n";  cout << "7 - очистка дерева\n";  cout << "8 - количество левых дочерних вершин\n";  cout << "9 - 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;  cout << "Введите слово: "; cin >> s;  insertElem(Root, key, s); break;  case 3: cout << "\nВведите ключ: "; cin >> key;  rc = search(Root, key);  cout << "Найденное слово= ";  puts(rc->text); 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 букву: "; cin >> letter;  n = count(Root, letter);  cout << "Количество слов, начинающихся с буквы " << letter;  cout << " равно " << n << endl; break;  case 7: delAll(Root); break;  case 8: numLeft = task(Root);  cout << "Количество левых дочерних вершин = " << numLeft << endl;  break;  case 9: exit(0);  }  }  }  Tree\* makeTree(Tree\* Root) //Создание дерева  {  int key; char s[20]; // для ключа и строки  cout << "Конец ввода - отрицательное число\n\n";  if (Root == NULL) // если дерево не создано  {  cout << "Введите ключ корня: "; cin >> key;  cout << "Введите слово корня: "; cin >> s;  Root = list(key, s); // установка указателя на корень  }  while (1) //добавление элементов  {  cout << "\nВведите ключ: "; cin >> key;  if (key < 0) break; //признак выхода (ключ < 0)  cout << "Введите слово: "; cin >> s;  insertElem(Root, key, s);  }  return Root;  }  Tree\* list(int i, char\* s) //Создание нового элемента  {  Tree\* t = new Tree[sizeof(Tree)]; // выделение памяти  t->key = i; // ключ  for (i = 0; i < 5; i++)  \*((t->text) + i) = \*(s + i); // создание  t->Left = t->Right = NULL; // правая и левая ветви  return t;  }  Tree\* insertElem(Tree\* t, int key, char\* s) //Добавление нового элемента  {  Tree\* Prev = NULL; // 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, s); //создается новый узел  if (key < Prev->key) // если ключ меньше  Prev->Left = t; // то помещается на левую ветвь  else  Prev->Right = t; // иначе 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; // переход на левую ветвь  else  if (Del->Left == NULL)  R = Del->Right; // или на правую  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(Tree\* t, char letter) //Подсчет количества букв  {  if (t)  {  count(t->Right, letter);  if (\*(t->text) == letter)  c++; // увеличиваем счетчик  count(t->Left, letter);  }  return c;  }  void 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 << ' ';  puts(t->text);  view(t->Left, level + 1); //вывод левого поддерева  }  }  void delAll(Tree\* t) //Очистка дерева  {  if (t != NULL)  {  delAll(t->Left);  delAll(t->Right);  delete t;  }  }  int task(Tree\* tree)  {  if (tree->Right != NULL)  {  task(tree->Right);  }  if (tree->Left != NULL)  {  task(tree->Left);  sum++; // подсчет количества листьев левой вершины  }  return sum;  } |
| Консоль отладки |
|  |

Вариант 7

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

|  |
| --- |
| Исходный код |
| // Дан указатель p1 на корень непустого дерева. Написать функцию вывода количества листьев данного дерева.  #include <iostream>  using namespace std;  struct Tree //дерево  {  int key; //ключ  char text[5]; //текст - не более 4 букв  Tree\* Left, \* Right; // укащатели на левую и правую ветви  };  int sum = 0, result3 = 0, counter2 = 0;  Tree\* makeTree(Tree\* Root); //Создание дерева  Tree\* list(int i, char\* s); //Создание нового элемента  Tree\* insertElem(Tree\* Root, int key, char\* s); //Добавление нового элемента  Tree\* search(Tree\* n, int key); //Поиск элемента по ключу  Tree\* delet(Tree\* Root, int key); //Удаление элемента по ключу  void view(Tree\* t, int level); //Вывод дерева  int count(Tree\* t, char letter); //Подсчет количества слов  void delAll(Tree\* t); //Очистка дерева  int task(Tree\* tree); // Подсчёт листьев  int c = 0; //количество слов  Tree\* Root = NULL; //указатель корня  void main()  {  setlocale(0, "Russian");  int key, choice, n, numLeft = 0;  Tree\* rc; char s[5], letter;  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 букв\n";  cout << "7 - очистка дерева\n";  cout << "8 - количество листьев\n";  cout << "9 - 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;  cout << "Введите слово: "; cin >> s;  insertElem(Root, key, s); break;  case 3: cout << "\nВведите ключ: "; cin >> key;  rc = search(Root, key);  cout << "Найденное слово= ";  puts(rc->text); 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 букву: "; cin >> letter;  n = count(Root, letter);  cout << "Количество слов, начинающихся с буквы " << letter;  cout << " равно " << n << endl; break;  case 7: delAll(Root); break;  case 8: numLeft = task(Root);  cout << "Количество правых дочерних вершин = " << numLeft << endl;  break;  case 9: exit(0);  }  }  }  Tree\* makeTree(Tree\* Root) //Создание дерева  {  int key; char s[20]; // для ключа и строки  cout << "Конец ввода - отрицательное число\n\n";  if (Root == NULL) // если дерево не создано  {  cout << "Введите ключ корня: "; cin >> key;  cout << "Введите слово корня: "; cin >> s;  Root = list(key, s); // установка указателя на корень  }  while (1) //добавление элементов  {  cout << "\nВведите ключ: "; cin >> key;  if (key < 0) break; //признак выхода (ключ < 0)  cout << "Введите слово: "; cin >> s;  insertElem(Root, key, s);  }  return Root;  }  Tree\* list(int i, char\* s) //Создание нового элемента  {  Tree\* t = new Tree[sizeof(Tree)]; // выделение памяти  t->key = i; // ключ  for (i = 0; i < 5; i++)  \*((t->text) + i) = \*(s + i); // создание  t->Left = t->Right = NULL; // правая и левая ветви  return t;  }  Tree\* insertElem(Tree\* t, int key, char\* s) //Добавление нового элемента  {  Tree\* Prev = NULL; // 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, s); //создается новый узел  if (key < Prev->key) // если ключ меньше  Prev->Left = t; // то помещается на левую ветвь  else  Prev->Right = t; // иначе 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; // переход на левую ветвь  else  if (Del->Left == NULL)  R = Del->Right; // или на правую  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(Tree\* t, char letter) //Подсчет количества букв  {  if (t)  {  count(t->Right, letter);  if (\*(t->text) == letter)  c++; // увеличиваем счетчик  count(t->Left, letter);  }  return c;  }  void 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 << ' ';  puts(t->text);  view(t->Left, level + 1); //вывод левого поддерева  }  }  void delAll(Tree\* t) //Очистка дерева  {  if (t != NULL)  {  delAll(t->Left);  delAll(t->Right);  delete t;  }  }  int task(Tree\* tree)  {  if (tree->Right != NULL)  {  task(tree->Right);  sum++; // подсчет количества листьев правой вершины  }  if (tree->Left != NULL)  {  task(tree->Left);  sum++; // подсчет количества листьев левой вершины  }  return sum;  } |
| Консоль отладки |
|  |

Вариант 12

Дан указатель **p1** на корень непустого дерева. Написать функцию определения количества узлов с четными ключами

|  |
| --- |
| Исходный код |
| // Дан указатель p1 на корень непустого дерева. Написать функцию определения количества узлов с четными ключами.  #include <iostream>  using namespace std;  struct Tree //дерево  {  int key; //ключ  char text[5]; //текст - не более 4 букв  Tree\* Left, \* Right; // укащатели на левую и правую ветви  };  int sum = 1, result3 = 0, counter2 = 0, counter = 0;  Tree\* makeTree(Tree\* Root); //Создание дерева  Tree\* list(int i, char\* s); //Создание нового элемента  Tree\* insertElem(Tree\* Root, int key, char\* s); //Добавление нового элемента  Tree\* search(Tree\* n, int key); //Поиск элемента по ключу  Tree\* delet(Tree\* Root, int key); //Удаление элемента по ключу  void view(Tree\* t, int level); //Вывод дерева  int count(Tree\* t, char letter); //Подсчет количества слов  void delAll(Tree\* t); //Очистка дерева  int task(Tree\* tree, int level); // Подсчёт листьев  int c = 0; //количество слов  Tree\* Root = NULL; //указатель корня  void main()  {  setlocale(0, "Russian");  int key, choice, n, numLeft = 0;  Tree\* rc; char s[5], letter;  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 букв\n";  cout << "7 - очистка дерева\n";  cout << "8 - количество листьев\n";  cout << "9 - 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;  cout << "Введите слово: "; cin >> s;  insertElem(Root, key, s); break;  case 3: cout << "\nВведите ключ: "; cin >> key;  rc = search(Root, key);  cout << "Найденное слово= ";  puts(rc->text); 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 букву: "; cin >> letter;  n = count(Root, letter);  cout << "Количество слов, начинающихся с буквы " << letter;  cout << " равно " << n << endl; break;  case 7: delAll(Root); break;  case 8: cout << "Количество дочерних вершин = " << task(Root, 0) << endl; break;  case 9: exit(0);  }  }  }  Tree\* makeTree(Tree\* Root) //Создание дерева  {  int key; char s[20]; // для ключа и строки  cout << "Конец ввода - отрицательное число\n\n";  if (Root == NULL) // если дерево не создано  {  cout << "Введите ключ корня: "; cin >> key;  cout << "Введите слово корня: "; cin >> s;  Root = list(key, s); // установка указателя на корень  }  while (1) //добавление элементов  {  cout << "\nВведите ключ: "; cin >> key;  if (key < 0) break; //признак выхода (ключ < 0)  cout << "Введите слово: "; cin >> s;  insertElem(Root, key, s);  }  return Root;  }  Tree\* list(int i, char\* s) //Создание нового элемента  {  Tree\* t = new Tree[sizeof(Tree)]; // выделение памяти  t->key = i; // ключ  for (i = 0; i < 5; i++)  \*((t->text) + i) = \*(s + i); // создание  t->Left = t->Right = NULL; // правая и левая ветви  return t;  }  Tree\* insertElem(Tree\* t, int key, char\* s) //Добавление нового элемента  {  Tree\* Prev = NULL; // 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, s); //создается новый узел  if (key < Prev->key) // если ключ меньше  Prev->Left = t; // то помещается на левую ветвь  else  Prev->Right = t; // иначе 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; // переход на левую ветвь  else  if (Del->Left == NULL)  R = Del->Right; // или на правую  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(Tree\* t, char letter) //Подсчет количества букв  {  if (t)  {  count(t->Right, letter);  if (\*(t->text) == letter)  c++; // увеличиваем счетчик  count(t->Left, letter);  }  return c;  }  void 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 << ' ';  puts(t->text);  view(t->Left, level + 1); //вывод левого поддерева  }  }  void delAll(Tree\* t) //Очистка дерева  {  if (t != NULL)  {  delAll(t->Left);  delAll(t->Right);  delete t;  }  }  int task(Tree\* tr, int level)  {  if (tr)  {  task(tr->Right, level + 1); //вывод правого поддерева  if (tr->key % 2 == 0)  counter++;  task(tr->Left, level + 1); //вывод левого поддерева  }  return counter;  } |
| Консоль отладки |
|  |