**Лабораторная работа № 11. Бинарные деревья**

|  |  |  |
| --- | --- | --- |
| 1. Изучить работу с *бинарным деревом*, выполнив программу, записанную в правой части.  В программе осуществляется добавление элемента в дерево и вывод дерева на экран с поворотом на 90 градусов влево. | **#include <iostream>**  **#include<conio.h>**  **using namespace std;**  **struct Node**  **{ int data; //Информационное поле**  **Node \*left, \*right; //Указатели на левую и правую ветви дерева**  **};**  **Node\* tree = nullptr;**  **void insert(int a, Node \*\*t) //Добавление элемента a**  **{ if ((\*t)==NULL) //если дерева нет, то создается элемент**  **{ (\*t) = new Node;**  **(\*t)->data = a;**  **(\*t)->left = (\*t)->right = NULL;**  **return;**  **}**  **if (a > (\*t)->data) //дерево есть, если а больше текущего**  **insert(a, &(\*t)->right); //то элемент помещается вправо**  **else**  **insert(a, &(\*t)->left); //иначе - влево**  **}**  **void print(Node \*t, int u) //Вывод на экран**  **{ if (t == NULL) return;**  **else**  **{ print(t->left, ++u); //левое поддерево**  **for (int i = 0; i < u; ++i)**  **cout << "|";**  **cout << t->data << endl;**  **u--;**  **}**  **print(t->right, ++u); // правое поддерево**  **}**  **void main()**  **{ setlocale(LC\_CTYPE, "Russian");**  **int count, temp;**  **cout << "Введите количество элементов "; cin >> count;**  **for (int i = 0; i<count; ++i)**  **{ cout << "Введите число "; cin >> temp;**  **insert(temp, &tree);**  **}**  **cout << "ваше дерево\n";**  **print(tree, 0);**  **\_getch();**  **}** |  |
| 2. В правой части приведена программа, которая осуществляет построение бинарного дерева, каждый элемент которого состоит из ключа (целое число) и слова размером не более 4 символов. В программе осуществляется добавление элемента, удаление элемента, поиск элемента по ключу, вывод дерева на экран, очистка дерева.  При вводе информации признак окончания − ввод отрицательного ключа.  Выполнить программу и написать комментарии к операторам. | Пример. Определить количество записей в бинарном дереве, начинающихся с введенной с клавиатуры буквы.  **#include <iostream>**  **using namespace std;**  **struct Tree //дерево**  **{ int key; //ключ**  **char text[5]; //текст - не более 4 букв**  **Tree \*Left, \*Right;**  **};**  **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**  **int view(Tree \*t, int level); //Вывод дерева**  **int count(Tree \*t, char letter); //Подсчет количества слов**  **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;**  **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 - 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: exit(0);**  **}**  **}**  **}**  **Tree\* makeTree(Tree \*Root) //Создание дерева**  **{ int key; char s[5];**  **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)];**  **t -> key = i;**  **for(i = 0; i < 5; i++)**  **\*((t -> text)+i) = \*(s+i);**  **t -> Left = t -> Right = NULL;**  **return t;**  **}**  **Tree\* insertElem(Tree \*t, int key, char \*s) //Добавление нового элемента**  **{ Tree \*Prev; // 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 ) // 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(Tree \*t, char letter) //Подсчет количества слов**  **{ if (t)**  **{ count(t -> Right, letter);**  **if(\*(t->text) == letter)**  **c++;**  **count(t -> Left, letter);**  **}**  **return c;**  **}**  **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<<' ';**  **puts(t->text);**  **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;**  **}** |  |
| 3. Разработать программу работы с ***бинарным деревом поиска***, в которую включить основные функции манипуляции данными и функцию в соответствии со своим вариантом из таблицы, представленной ниже. | | |
| Вариант 10  Дан указатель **p1** на корень непустого дерева. Написать функцию вывода вывода суммы значений всех вершин данного дерева.  Вариант 16  Дан указатель p1 на корень непустого дерева. Написать функцию вывода суммы значений всех листьев данного дерева.  Вариант 1  Вершина бинарного дерева содержит ключ, три целых числа и два указателя на потомков. Написать функцию удаления вершины с минимальной суммой трех целых значений узла.  Вариант 2  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 {  int key; // ключ  Tree\* Left, \* Right;  };  Tree\* makeTree(Tree\* Root); // Создание дерева  Tree\* list(int i); // Создание нового элемента  Tree\* insertElem(Tree\* Root, int key); // 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  void delAll(Tree\* t); // Очистка дерева  int sumTreeNodes(Tree\* p1);  int sumLeafs(Tree\* p1);  int countRightLeaves(Tree\* p1);  Tree\* Root = NULL; // указатель на корень  int main() {  setlocale(0, "Russian");  int key, choice;  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 - очистка дерева\n";  cout << "7 - выход\n";  cout << "8 - сумма дерева\n";  cout << "9 - сумма листьев дерева\n";  cout << "10 - вывод количества листьев дерева, которые являются правыми дочерними вершинами\n";  cout << "ваш выбор?\n";  cin >> choice;  cout << "\n";  switch (choice) {  case 1: {  Root = makeTree(Root);  break;  }  case 2: {  cout << "\nВведите ключ: ";  cin >> key;  insertElem(Root, key);  break;  }  case 3: {  cout << "\nВведите ключ: ";  cin >> key;  Tree\* rc = search(Root, key);  if (rc)  cout << "Найден элемент с ключом: " << rc->key << endl;  break;  }  case 4: {  cout << "\nВведите удаляемый ключ: ";  cin >> key;  Root = delet(Root, key);  break;  }  case 5: {  if (Root) {  cout << "Дерево повернуто на 90 град. влево" << endl;  view(Root, 0);  }  else  cout << "Дерево пустое\n";  break;  }  case 6: {  delAll(Root);  Root = NULL;  break;  }  case 7:  exit(0);  case 8: {  if (Root) {  int totalSum = sumTreeNodes(Root);  cout << "Сумма всех вершин дерева: " << totalSum << endl;  }  else {  cout << "Дерево пустое\n";  }  break;  }  case 9: {  if (Root) {  int totalSum = sumLeafs(Root);  cout << "Сумма всех листьев дерева: " << totalSum << endl;  }  else {  cout << "Дерево пустое\n";  }  }  case 10: {  if (Root) {  int rightLeavesCount = countRightLeaves(Root);  cout << "Количество правых листьев: " << rightLeavesCount << endl;  }  else {  cout << "Дерево пустое\n";  }  break;  }  }  }  }  Tree\* makeTree(Tree\* Root) {  int key;  cout << "Конец ввода - отрицательное число\n\n";  if (Root == NULL) {  cout << "Введите ключ корня: ";  cin >> key;  Root = list(key);  }  while (1) {  cout << "\nВведите ключ: ";  cin >> key;  if (key < 0) break;  insertElem(Root, key);  }  return Root;  }  Tree\* list(int i) {  Tree\* t = new Tree;  t->key = i;  t->Left = t->Right = NULL;  return t;  }  Tree\* insertElem(Tree\* t, int key) {  Tree\* Prev = nullptr;  int find = 0;  while (t && !find) {  Prev = t;  if (key == t->key)  find = 1;  else if (key < t->key)  t = t->Left;  else  t = t->Right;  }  if (!find) {  t = list(key);  if (key < Prev->key)  Prev->Left = t;  else  Prev->Right = t;  }  return t;  }  Tree\* delet(Tree\* Root, int key) {  Tree\* Del, \* Prev\_Del, \* R, \* Prev\_R;  Del = Root;  Prev\_Del = NULL;  while (Del != NULL && Del->key != key) {  Prev\_Del = Del;  if (Del->key > key)  Del = Del->Left;  else  Del = Del->Right;  }  if (Del == NULL) {  cout << "\nНет такого ключа";  return Root;  }  if (Del->Right == NULL)  R = Del->Left;  else if (Del->Left == NULL)  R = Del->Right;  else {  Prev\_R = Del;  R = Del->Left;  while (R->Right != NULL) {  Prev\_R = R;  R = R->Right;  }  if (Prev\_R == Del)  R->Right = Del->Right;  else {  R->Right = Del->Right;  Prev\_R->Right = R->Left;  R->Left = Prev\_R;  }  }  if (Del == Root)  Root = R;  else if (Del->key < Prev\_Del->key)  Prev\_Del->Left = R;  else  Prev\_Del->Right = R;  cout << "\nУдален элемент с ключом " << Del->key << endl;  delete Del;  return Root;  }  Tree\* search(Tree\* n, int key) {  if (n == NULL || n->key == key)  return n;  if (key < n->key)  return search(n->Left, key);  else  return search(n->Right, key);  }  void view(Tree\* t, int level) {  if (t) {  view(t->Right, level + 1);  for (int i = 0; i < level; i++)  cout << " ";  cout << t->key << endl;  view(t->Left, level + 1);  }  }  void delAll(Tree\* t) {  if (t != NULL) {  delAll(t->Left);  delAll(t->Right);  delete t;  }  }  int sumTreeNodes(Tree\* p1) {  if (p1 == nullptr) {  return 0;  }  return p1->key + sumTreeNodes(p1->Left) + sumTreeNodes(p1->Right);  }  int sumLeafs(Tree\* p1) {  if (p1->Left == nullptr && p1->Right == nullptr) {  return p1->key;  }  return sumLeafs(p1->Left) + sumLeafs(p1->Right);  }  int countRightLeaves(Tree\* p1) {  if (p1 == nullptr) {  return 0;  }  int count = 0;  if (p1->Right != nullptr && p1->Right->Left == nullptr && p1->Right->Right == nullptr) {  count++;  }  count += countRightLeaves(p1->Left);  count += countRightLeaves(p1->Right);  return count;  }  Вариант 1  #include <iostream>  #include <climits>  using namespace std;  struct Tree {  int key; // ключ  int a, b, c; // три целых числа  Tree\* Left, \* Right;  };  // Прототипы функций  Tree\* makeTree(Tree\* Root);  Tree\* list(int i, int a, int b, int c);  Tree\* insertElem(Tree\* Root, int key, int a, int b, int c);  Tree\* search(Tree\* n, int key);  Tree\* delet(Tree\* Root, int key);  void view(Tree\* t, int level);  void delAll(Tree\* t);  Tree\* findMinSumNode(Tree\* root);  Tree\* deleteMinSumNode(Tree\* root);  Tree\* Root = NULL;  int main() {  setlocale(0, "Russian");  int key, choice, a, b, c;  for (;;) {  cout << "1 - создание дерева\n";  cout << "2 - добавление элемента\n";  cout << "3 - поиск по ключу\n";  cout << "4 - удаление элемента по ключу\n";  cout << "5 - вывод дерева\n";  cout << "6 - очистка дерева\n";  cout << "7 - выход\n";  cout << "8 - удалить вершину с минимальной суммой (a+b+c)\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 три числа через пробел: ";  cin >> a >> b >> c;  insertElem(Root, key, a, b, c);  break;  }  case 3: {  cout << "\nВведите ключ: ";  cin >> key;  Tree\* rc = search(Root, key);  if (rc)  cout << "Найден элемент: ключ=" << rc->key << ", значения="  << rc->a << "," << rc->b << "," << rc->c << endl;  break;  }  case 4: {  cout << "\nВведите удаляемый ключ: ";  cin >> key;  Root = delet(Root, key);  break;  }  case 5: {  if (Root) {  cout << "Дерево повернуто на 90 град. влево" << endl;  view(Root, 0);  }  else  cout << "Дерево пустое\n";  break;  }  case 6: {  delAll(Root);  Root = NULL;  break;  }  case 7:  exit(0);  case 8: {  if (Root) {  Root = deleteMinSumNode(Root);  cout << "Вершина с минимальной суммой удалена\n";  }  else {  cout << "Дерево пустое\n";  }  break;  }  }  }  }  Tree\* makeTree(Tree\* Root) {  int key, a, b, c;  cout << "Конец ввода - отрицательное число\n\n";  if (Root == NULL) {  cout << "Введите ключ корня: ";  cin >> key;  cout << "Введите три числа через пробел: ";  cin >> a >> b >> c;  Root = list(key, a, b, c);  }  while (1) {  cout << "\nВведите ключ: ";  cin >> key;  if (key < 0) break;  cout << "Введите три числа через пробел: ";  cin >> a >> b >> c;  insertElem(Root, key, a, b, c);  }  return Root;  }  Tree\* list(int i, int a, int b, int c) {  Tree\* t = new Tree;  t->key = i;  t->a = a;  t->b = b;  t->c = c;  t->Left = t->Right = NULL;  return t;  }  Tree\* insertElem(Tree\* t, int key, int a, int b, int c) {  Tree\* Prev = nullptr;  int find = 0;  while (t && !find) {  Prev = t;  if (key == t->key)  find = 1;  else if (key < t->key)  t = t->Left;  else  t = t->Right;  }  if (!find) {  t = list(key, a, b, c);  if (key < Prev->key)  Prev->Left = t;  else  Prev->Right = t;  }  return t;  }  Tree\* delet(Tree\* Root, int key) {  Tree\* Del, \* Prev\_Del, \* R, \* Prev\_R;  Del = Root;  Prev\_Del = NULL;  while (Del != NULL && Del->key != key) {  Prev\_Del = Del;  if (Del->key > key)  Del = Del->Left;  else  Del = Del->Right;  }  if (Del == NULL) {  cout << "\nНет такого ключа";  return Root;  }  if (Del->Right == NULL)  R = Del->Left;  else if (Del->Left == NULL)  R = Del->Right;  else {  Prev\_R = Del;  R = Del->Left;  while (R->Right != NULL) {  Prev\_R = R;  R = R->Right;  }  if (Prev\_R == Del)  R->Right = Del->Right;  else {  R->Right = Del->Right;  Prev\_R->Right = R->Left;  R->Left = Prev\_R;  }  }  if (Del == Root)  Root = R;  else if (Del->key < Prev\_Del->key)  Prev\_Del->Left = R;  else  Prev\_Del->Right = R;  cout << "\nУдален элемент с ключом " << Del->key << endl;  delete Del;  return Root;  }  Tree\* search(Tree\* n, int key) {  if (n == NULL || n->key == key)  return n;  if (key < n->key)  return search(n->Left, key);  else  return search(n->Right, key);  }  void view(Tree\* t, int level) {  if (t) {  view(t->Right, level + 1);  for (int i = 0; i < level; i++)  cout << " ";  cout << t->key << " (" << t->a << "," << t->b << "," << t->c << ")" << endl;  view(t->Left, level + 1);  }  }  void delAll(Tree\* t) {  if (t != NULL) {  delAll(t->Left);  delAll(t->Right);  delete t;  }  }  Tree\* findMinSumNode(Tree\* root) {  if (root == NULL) return NULL;  Tree\* minNode = root;  int minSum = root->a + root->b + root->c;  Tree\* leftMin = findMinSumNode(root->Left);  if (leftMin != NULL) {  int leftSum = leftMin->a + leftMin->b + leftMin->c;  if (leftSum < minSum) {  minSum = leftSum;  minNode = leftMin;  }  }  Tree\* rightMin = findMinSumNode(root->Right);  if (rightMin != NULL) {  int rightSum = rightMin->a + rightMin->b + rightMin->c;  if (rightSum < minSum) {  minSum = rightSum;  minNode = rightMin;  }  }  return minNode;  }  Tree\* deleteMinSumNode(Tree\* root) {  if (root == NULL) return NULL;  Tree\* minNode = findMinSumNode(root);  if (minNode != NULL) {  return delet(root, minNode->key);  }  return root;  } |  |