Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Лабораторная работа № 11

По дисциплине «Основы алгоритмизации и программирования»

На тему «Бинарные деревья»

Выполнил:

Рауба Арсений

Студент 1 курса 8 группы

Преподаватель: асс. Андронова М.В.

Минск, 2024

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 №9

Дан указатель p1 на корень непустого дерева и число k. Написать функцию вывода количества вершин дерева, значение которых равно k.

#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\* 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

void insert(int a,string s, Tree\*\* t);

int c = 0; //количество слов

Tree\* Root = NULL; //указатель на корень

int k;

void main(int counter)

{

int k;

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;

cout << "Введите k: ";

cin >> k;

insert(key, s, &Root);

cout <<"Количество вершин ==K: "<< counter<<endl;

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;

insert(key, s, &Root);

}

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;

}

void insert(int a, string s, Tree\*\* t) //Добавление элемента a

{

int counter = 0;

if ((\*t) == NULL) //если дерева нет, то создается элемент

{

(\*t) = new Tree;

(\*t)->key = a;

(\*t)->Left = (\*t)->Right = NULL;

return;

}

if (a > (\*t)->key) //дерево есть, если а больше текущего

insert(a,s, &(\*t)->Right); //то элемент помещается вправо

else

insert(a,s, &(\*t)->Left); //иначе - влево

if ((\*t)->key == k) {

counter += 1;

}

}

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);

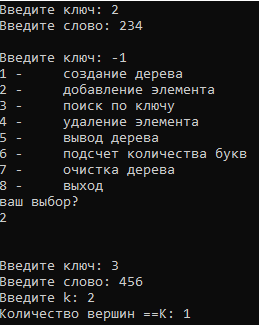
delete t;

return true;

}

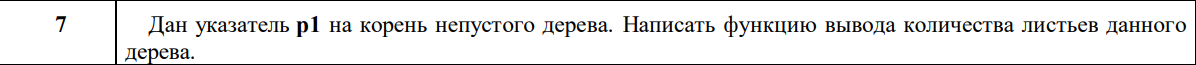
return false;

}



**Дополнительные задания.**

**Вариант 7**

****

#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

int countno(Tree\* p, int level, int &counter); //Вывод кол-ва листьев дерева

Tree\* Root = NULL; //указатель на корень

void main()

{

setlocale(0, "Russian");

int key, choice, n;

int counter = 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 - подсчет кол-ва узлов с четными ключами\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 << "Кол-во листьев дерева " << countno(Root, 0, counter) - 1 << endl;

break;

case 9: 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;

}

int countno(Tree\* p, int level, int &counter) //Вывод кол-ва листьев дерева

{

if (p)

{

counter++;

countno(p->Right, level + 1, counter);

countno(p->Left, level + 1, counter);

return counter;

}

return counter;

}

bool delAll(Tree \* t) //Очистка дерева

{

if (t != NULL)

{

delAll(t->Left);

delAll(t->Right);

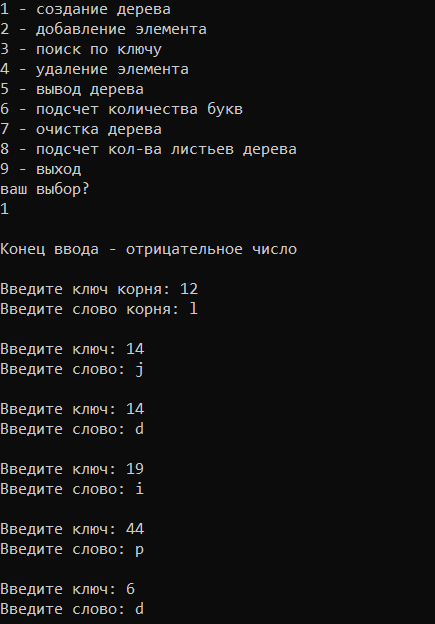
delete t;

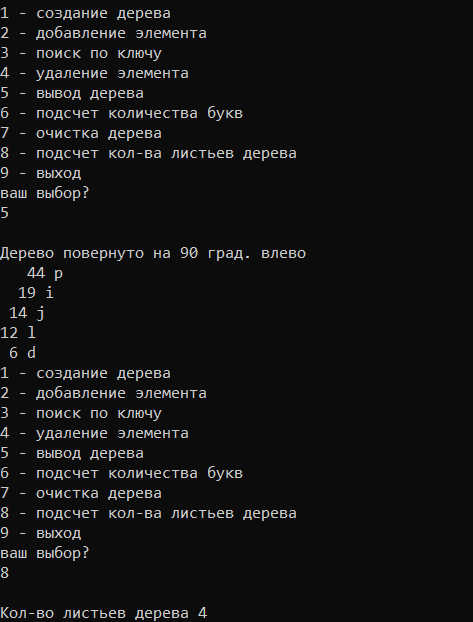
return true;

}

return false;

}

****

****

**Вариант 5**

****

#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

int sum(Tree\* t, int level, int &count); //Подсчет кол-ва левых вершин

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;

int counter = 0;

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 - подсчет кол-ва левых 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:

cout << "Кол-во левых вершин: " << sum(Root, 0, counter) << endl;

break;

case 9: 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;

}

int sum(Tree\* t, int level, int &count) //Подсчет кол-ва левых вершин

{

if (t)

{

int tm = t->key;

count++;

sum(t->Left, level + 1, count);

return count;

}

return count;

}

bool delAll(Tree \* t) //Очистка дерева

{

if (t != NULL)

{

delAll(t->Left);

delAll(t->Right);

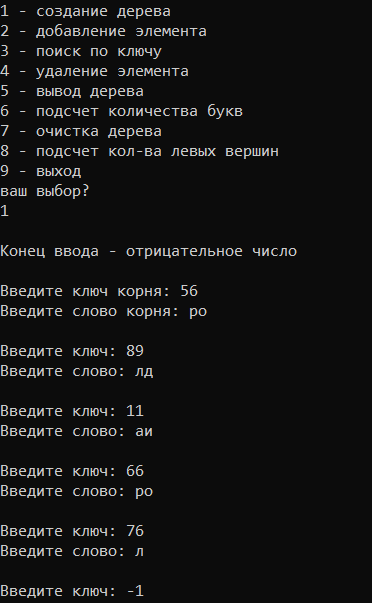
delete t;

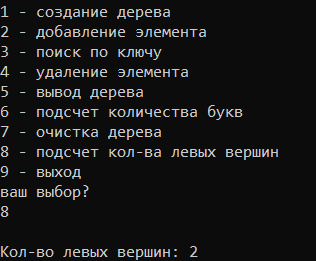
return true;

}

return false;

}

****

****

**Вариант 14**

****

#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

int avg(Tree\* t, int level, int& count, int& summa); //Подсчет кол-ва левых вершин

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;

int counter = 0, sum = 0;

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 - подсчет кол-ва левых 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:

cout << "Среднее: " << avg(Root, 0, counter, sum)/counter << endl;

break;

case 9: 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;

}

int avg(Tree\* t, int level, int &count, int &summa) //Подсчет среднего арифметического

{

if (t)

{

int tm = t->key;

count++;

summa += tm;

avg(t->Left, level + 1, count, summa);

avg(t->Right, level + 1, count, summa);

return summa;

}

return summa;

}

bool delAll(Tree \* t) //Очистка дерева

{

if (t != NULL)

{

delAll(t->Left);

delAll(t->Right);

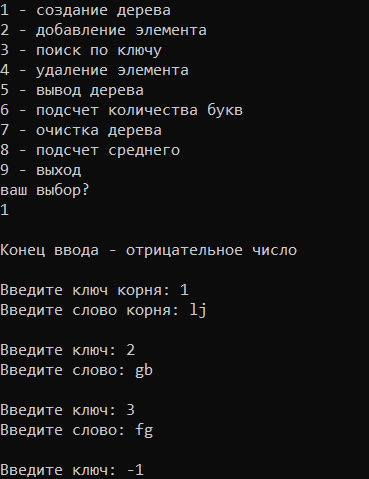
delete t;

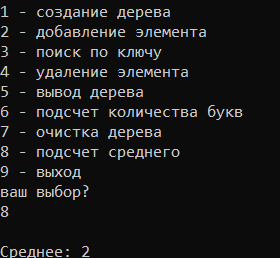
return true;

}

return false;

}

****

****