Банкузов Михаил 7 группа Лабораторная работа №11 Вариант 1



#include <iostream>

using namespace std;

struct Tree//дерево

{

int key;//ключ

int n1 = 0, n2 = 0, n3 = 0;

int sum = n1 + n2 + n3;

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, int n1, int n2, int n3);//Создание нового элемента

Tree\* insertElem(Tree\* Root, int key, int n1, int n2, int n3);//Добавление нового элемента

Tree\* delet(Tree\* Root); //Удаление элемента по ключу

void view(Tree\* t, int level);//Вывод дерева

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;

int n1, n2, n3;

for (;;)

{

cout << "1 - создание дерева\n";

cout << "2 - добавление элемента\n";

cout << "3 - удаление элемента\n";

cout << "4 - вывод дерева\n";

cout << "5 - очистка дерева\n";

cout << "6 - количество правых дочерних вершин\n";

cout << "7 - 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 значение полей: "; cin >> n1 >> n2 >> n3;

insertElem(Root, key, n1, n2, n3); break;

case 3:

Root = delet(Root); break;

case 4: if (Root->key >= 0)

{

cout << "Дерево повернуто на 90 град. влево" << endl;

view(Root, 0);

}

else cout << "Дерево пустое\n"; break;

case 5: delAll(Root); break;

case 6: numLeft = check\_left(Root);

cout << "Количество правых дочерних вершин = " << numLeft << endl;

break;

case 7: exit(0);

}

}

}

Tree\* makeTree(Tree\* Root)//Создание дерева

{

int key; int n1, n2, n3;

cout << "Конец ввода - отрицательное число\n\n";

if (Root == NULL)// если дерево не создано

{

cout << "Введите ключ корня: "; cin >> key;

cout << "Введите значения полей: "; cin >> n1 >> n2 >> n3;

Root = list(key, n1, n2, n3);// установка указателя на корень

}

while (1)//добавление элементов

{

cout << "\nВведите ключ: "; cin >> key;

if (key < 0) break;//признак выхода (ключ < 0)

cout << "Введите значения полей: "; cin >> n1 >> n2 >> n3;

insertElem(Root, key, n1, n2, n3);

}

return Root;

}

Tree\* list(int i, int n1, int n2, int n3)//Создание нового элемента

{

Tree\* t = new Tree[sizeof(Tree)];// выделение памяти

t->key = i;// ключ

t->n1 = n1; t->n2 = n2; t->n3 = n3;

t->sum = n1 + n2 + n3;

t->Left = t->Right = NULL;// правая и левая ветви

return t;

}

Tree\* insertElem(Tree\* t, int key, int n1, int n2, int n3)//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, n1, n2, n3);//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;

}

static void search\_min(Tree\* t, int& min\_sum, int& min\_key)

{

if (t)

{

search\_min(t->Right, min\_sum, min\_key); //вывод правого поддерева

if (t->sum < min\_sum) { min\_sum = t->sum; min\_key = t->key; }

search\_min(t->Left, min\_sum, min\_key); //вывод левого поддерева

}

}

Tree\* delet(Tree\* Root)

{

Tree\* Del, \* Prev\_Del, \* R, \* Prev\_R;

Del = Root;

int min\_sum = Del->sum;

int min\_key = Del->key;

search\_min(Root, min\_sum, min\_key);

Prev\_Del = NULL;

while (Del != NULL && Del->key != min\_key)//поиск элемента и его родителя

{

Prev\_Del = Del;

if (Del->key > min\_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;

}

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 << ' ';

cout << t->n1 << ':' << t->n2 << ':' << t->n3;

cout << endl;

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

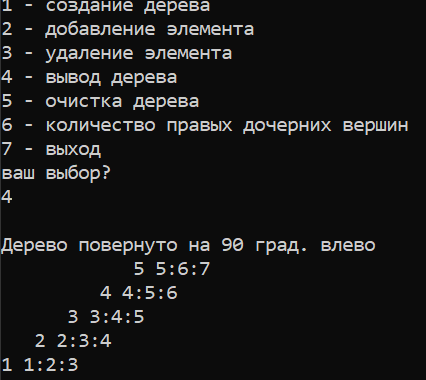
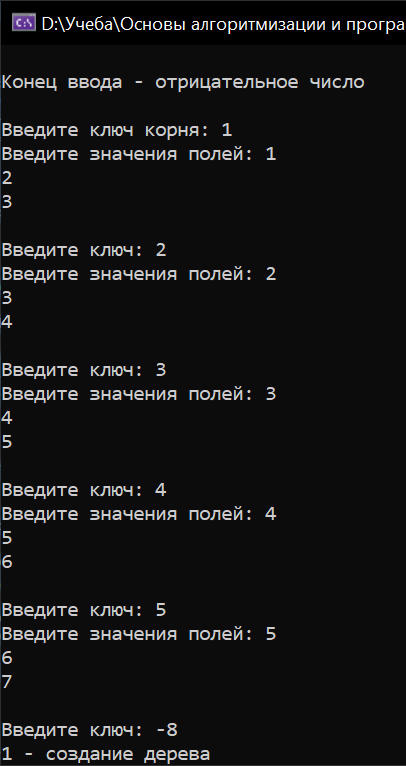
}

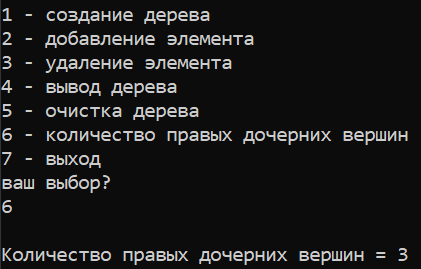
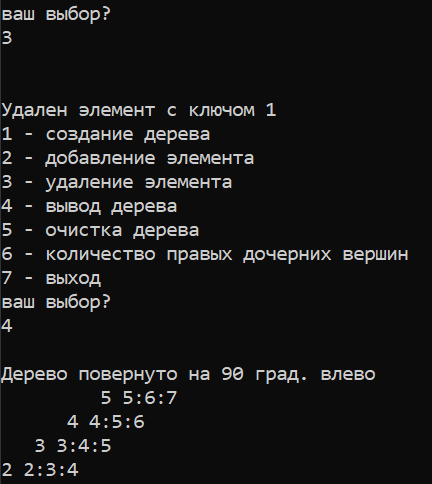
if (tree->Left != NULL)

check\_left(tree->Left);

return sum;

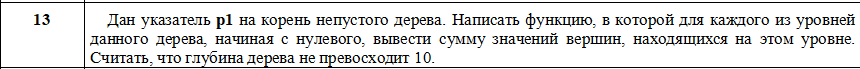
}





Дополнительные задания

Вариант 13



#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

void view(Tree\* t, int level);//Вывод дерева

int count(Tree\* t);//Подсчет количества листьев правой дочерней ветви

void delAll(Tree\* t);//Очистка дерева

int c = 0;//количество слов

Tree\* Root = NULL;//указатель корня

int 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 листьев правой дочерней ветви\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: n = count(Root);

n--;

cout << "Количество листьев правой дочерней ветви " << n << endl;

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 = new Tree[sizeof(Tree)]; // 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)//Подсчет количества листьев правой дочерней вершины

{

if (t)

{

count(t->Right);

c++;

}

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)

{

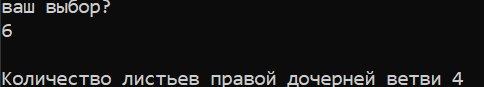
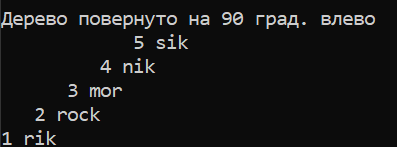
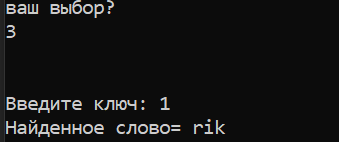
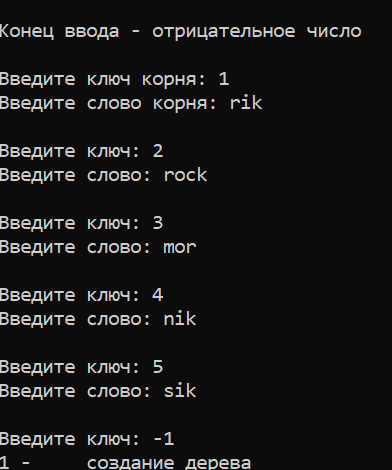
delAll(t->Left);

delAll(t->Right);

delete t;

}

}



Вариант 2



#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

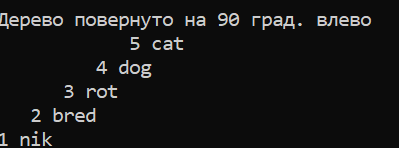
}

if (tree->Left != NULL)

check\_left(tree->Left);

return sum;

}





Вариант 7



#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;

}

