**Курс «Объектно-ориентированное программирование на C++»**

**Встреча №17**

Тема: Бинарное дерево

Домашнее задание

**Задание №1.** Создать класс для работы с бинарным деревом, содержащим англо-русский словарь. Каждый узел содержит счетчик обращений к слову в данном узле. В процессе эксплуатации словаря при каждом вызове слова в счетчик обращений добавляется единица.

Написать программу, которая:

a) обеспечивает начальный ввод словаря с конкретными значениями счетчика обращений;

b) позволяет отобразить слово и его перевод

с) позволяет добавить,заменить,удалить перевод слова

d) отображает топ-3 самых популярных слов (определяем популярность на основании счетчика обращений)

e) отображает топ-3 самых непопулярных слов (определяем непопулярность на основании счетчика обращений)

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<Windows.h>

#include <iostream>

#include<conio.h>

#include<time.h>

using namespace std;

class String

{

private:

char\*\* arr;

int\* look;

int k;

public:

String()

{

for (int i = 0; i < k; i++)

{

arr[i] = NULL;

}

look = NULL;

arr = NULL;

k = 0;

}

String(const String& obj)

{

look = new int[obj.k];

arr = new char\* [obj.k];

k = obj.k;

for (int i = 0; i < k; i++)

{

arr[i] = new char[strlen(obj.arr[i]) + 1];

strcpy(arr[i], obj.arr[i]);

look[i] = obj.look[i];

}

}

String& operator=(const String& obj)

{

Delete\_memory();

look = new int[obj.k];

arr = new char\* [obj.k];

k = obj.k;

for (int i = 0; i < k; i++)

{

arr[i] = new char[strlen(obj.arr[i]) + 1];

strcpy(arr[i], obj.arr[i]);

look[i] = obj.look[i];

}

return \*this;

}

~String()

{

for (int i = 0; i < k; i++)

{

if (arr[i] != NULL)

delete[] arr[i];

}

if (arr != NULL)

delete[] arr;

if (look != NULL)

delete look;

}

void Delete\_memory()

{

for (int i = 0; i < k; i++)

{

if (arr[i] != NULL)

delete[] arr[i];

}

if (arr != NULL)

delete[] arr;

if (look != NULL)

delete look;

}

void Print\_String()

{

for (int i = 0; i < k; i++)

{

cout << arr[i] << endl;

}

cout << "================================\n";

}

void String\_memory(int s)

{

arr = new char\* [s];

}

void Sort()

{

bool b;

do {

b = false;

for (int i = 0; i < k - 1; i++)

{

if (look[i] > look[i + 1])

{

int tmp2;

tmp2 = look[i];

look[i] = look[i + 1];

look[i + 1] = tmp2;

char\* temp = new char[strlen(arr[i]) + 1];

strcpy(temp, arr[i]);

if (arr[i] != NULL)

delete[] arr[i];

arr[i] = new char[strlen(arr[i + 1]) + 1];

strcpy(arr[i], arr[i + 1]);

if (arr[i + 1] != NULL)

delete[] arr[i + 1];

arr[i + 1] = new char[strlen(temp) + 1];

strcpy(arr[i + 1], temp);

b = true;

delete[] temp;

}

}

} while (b);

}

void Show\_first(int j = 3)

{

Sort();

for (int i = k - 1; i >= 0 && i >= k - j; i--)

{

cout << arr[i] << endl;

}

}

void Show\_last(int j = 3)

{

Sort();

for (int i = 0; i < k && i < j; i++)

{

cout << arr[i] << endl;

}

}

String Add\_String(char\* str, int look\_)

{

String tmp;

tmp.look = new int[k + 1];

tmp.String\_memory(k + 1);

tmp.k = k + 1;

for (int i = 0; i < k; i++)

{

tmp.arr[i] = new char[strlen(arr[i]) + 1];

strcpy(tmp.arr[i], arr[i]);

tmp.look[i] = look[i];

}

tmp.arr[k] = new char[strlen(str) + 1];

strcpy(tmp.arr[k], str);

tmp.look[k] = look\_;

return tmp;

}

void Delete\_String(char\* str)

{

char\*\* tmp = new char\* [k - 1];

int\* tmp2 = new int[k - 1];

int index = 0;

for (int i = 0; i < k; i++)

{

if (strcmp(arr[i], str) == 0)

{

index = i;

break;

}

}

for (int i = 0; i < index; i++)

{

tmp[i] = new char[strlen(arr[i]) + 1];

strcpy(tmp[i], arr[i]);

tmp2[i] = look[i];

}

int j = index + 1;

for (int i = index; i < k - 1; i++)

{

tmp[i] = new char[strlen(arr[j]) + 1];

strcpy(tmp[i], arr[j]);

tmp2[i] = look[j];

j++;

}

Delete\_memory();

arr = tmp;

look = tmp2;

}

};

struct Elem

{

char\* eng\_word;

char\* ukr\_translation;

int count;

Elem\* left;

Elem\* right;

Elem\* parent;

void Add(const char\* eng, const char\* ukr, int count)

{

this->eng\_word = new char[strlen(eng) + 1];

this->ukr\_translation = new char[strlen(ukr) + 1];

strcpy(this->eng\_word, eng);

strcpy(this->ukr\_translation, ukr);

this->count = count;

}

void Show()

{

cout << eng\_word << " <-> " << ukr\_translation << endl;

cout << "Шукали це слово: " << count << endl;

cout << "---------------------------------------\n";

}

};

class Tree

{

private:

String top;

Elem\* root;

public:

Tree();

~Tree();

void Print(Elem\* node);

Elem\* Search(Elem\* node, const char\* key);

Elem\* Min(Elem\* node);

Elem\* Max(Elem\* node);

Elem\* Next(Elem\* node);

Elem\* Prev(Elem\* node);

void Insert(Elem\* new\_el);

void Del(Elem\* el = 0);

Elem\* GetRoot();

Elem\* Redact(const char\* key, const char\* new\_translation);

Elem\* Delete\_Translation(const char\* key);

Elem\* Add\_Translation(const char\* key, const char\* new\_translation);

void Show\_Elem(Elem\* node);

void Show\_Top3\_Popular(int k = 3);

void Show\_Top\_Least(int k = 3);

};

void Tree::Show\_Top3\_Popular(int k )

{

top.Show\_first(k);

}

void Tree::Show\_Top\_Least(int k )

{

top.Show\_last(k);

}

Elem\* Tree::Add\_Translation(const char\* key, const char\* new\_translation)

{

Elem\* node = root;

while (node != NULL && (strcmp(key, node->eng\_word) != 0))

{

if (strcmp(key, node->eng\_word) == -1)

node = node->left;

else

node = node->right;

}

if (node == NULL)

return node;

else

{

if (node->ukr\_translation != NULL)

{

node->ukr\_translation = (char\*)realloc(node->ukr\_translation, sizeof(char) \* (strlen(node->ukr\_translation) + 1 + 2 + strlen(new\_translation)));

strcat(node->ukr\_translation, "; ");

strcat(node->ukr\_translation, new\_translation);

}

else

{

node->ukr\_translation = new char[strlen(new\_translation) + 1];

strcpy(node->ukr\_translation, new\_translation);

}

return node;

}

}

void Tree::Show\_Elem(Elem\* node)

{

cout << "---------------------------------------\n";

cout << node->eng\_word << " <-> ";

if (node->ukr\_translation != NULL)

cout << node->ukr\_translation << endl;

else

cout << "переклад відсутній\n";

cout << "---------------------------------------\n";

}

Elem\* Tree::Redact(const char\* key, const char\* new\_translation)

{

Elem\* node = root;

while (node != NULL && (strcmp(key, node->eng\_word) != 0))

{

if (strcmp(key, node->eng\_word) == -1)

node = node->left;

else

node = node->right;

}

if (node == NULL)

return node;

else /\*(strcmp(key, node->eng\_word) == 0)\*/

{

if (strlen(node->ukr\_translation) != 0)

delete[]node->ukr\_translation;

node->ukr\_translation = new char[strlen(new\_translation) + 1];

strcpy(node->ukr\_translation, new\_translation);

return node;

}

/\*else

return NULL;\*/

}

Elem\* Tree::Delete\_Translation(const char\* key)

{

Elem\* node = root;

while (node != NULL && (strcmp(key, node->eng\_word) != 0))

{

if (strcmp(key, node->eng\_word) == -1)

node = node->left;

else

node = node->right;

}

if (node == NULL)

return node;

else /\*(strcmp(key, node->eng\_word) == 0)\*/

{

if (strlen(node->ukr\_translation) != 0)

delete[]node->ukr\_translation;

node->ukr\_translation = NULL;

return node;

}

}

Elem\* Tree::GetRoot()

{

return root;

}

Tree::Tree()

{

root = NULL;

}

Tree::~Tree()

{

//top.Delete\_memory();

Del();

}

Elem\* Tree::Search(Elem\* node, const char\* key)

{

while (node != NULL && (strcmp(key, node->eng\_word) != 0))

{

if (strcmp(key, node->eng\_word) == -1)

node = node->left;

else

node = node->right;

}

if (strcmp(key, node->eng\_word) == 0)

node->count++;

return node;

}

void Tree::Print(Elem\* node)

{

if (node != NULL)

{

Print(node->left);

cout << node->eng\_word << " <-> ";

if (node->ukr\_translation != NULL)

cout << node->ukr\_translation << endl;

else

cout << "переклад відсутній\n";

cout << "---------------------------------------\n";

Print(node->right);

}

}

void Tree::Insert(Elem\* new\_el)

{

top = top.Add\_String(new\_el->eng\_word, new\_el->count);

//нащадків в нового елемента не існує

new\_el->left = NULL;

new\_el->right = NULL;

Elem\* Node = root;

Elem\* prev\_el = Node;

//пошук місця для додавання в нього нового елемента

while (Node != NULL)

{

prev\_el = Node;

if (strcmp(new\_el->eng\_word, Node->eng\_word) == -1)

Node = Node->left;

else

Node = Node->right;

}

//Node після циклу НУЛЛ, а prev\_el - це елемент, де була Node до того, як "вийшла" за межі бінарного дерева

new\_el->parent = prev\_el;

if (prev\_el == NULL)

root = new\_el;

else

{

if (strcmp(new\_el->eng\_word, prev\_el->eng\_word) == -1)

prev\_el->left = new\_el;

else

prev\_el->right = new\_el;

}

}

Elem\* Tree::Next(Elem\* node)

{

Elem\* tmp = NULL;

if (node != NULL)

{

if (node->right != NULL)

return Min(node->right);

tmp = node->parent;

while (tmp != NULL && node == tmp->right)

{

node = tmp;

tmp = tmp->parent;

}

}

return tmp;

}

Elem\* Tree::Prev(Elem\* node)

{

Elem\* tmp = NULL;

if (node != NULL)

{

if (node->left != NULL)

return Max(node->left);

tmp = node->parent;

while (tmp != NULL && node == tmp->left)

{

node = tmp;

tmp = tmp->parent;

}

}

return tmp;

}

Elem\* Tree::Min(Elem\* node)

{

if (node != NULL)

{

while (node->left != NULL)

{

node = node->left;

}

}

return node;

}

Elem\* Tree::Max(Elem\* node)

{

if (node != NULL)

{

while (node->right != NULL)

{

node = node->right;

}

}

return node;

}

void Tree::Del(Elem\* el)

{

if (el != NULL)//видаляється певний конкретний елемент

{

Elem\* node, \* tmp;

//не два нащадки

if (el->left == NULL || el->right == 0)

tmp = el;

else

tmp = Next(el);

if (tmp->left != NULL)

node = tmp->left;

else

node = tmp->right;

if (node != NULL)

node->parent = tmp->parent;

//якщо видаляється кореневий елемент

if (tmp->parent == NULL)

root = node;

else if (tmp == tmp->parent->left) // чи зліва від батьківського

tmp->parent->left = node;

else

tmp->parent->right = node;

if (tmp->ukr\_translation != NULL)

{

if (tmp != el)

{

if (strlen(el->eng\_word) != 0)//1111111111111111111111111111

delete[]el->eng\_word;

el->eng\_word = new char[strlen(tmp->eng\_word) + 1];

strcpy(el->eng\_word, tmp->eng\_word);

if (el->ukr\_translation != NULL)

delete[]el->ukr\_translation;

el->ukr\_translation = new char[strlen(tmp->ukr\_translation) + 1];

strcpy(el->ukr\_translation, tmp->ukr\_translation);

el->count = tmp->count;

}

}

if (tmp->eng\_word != NULL)

delete[]tmp->eng\_word;

if (tmp->ukr\_translation != NULL)

delete[]tmp->ukr\_translation;

delete tmp;

}

else // видалення всього дерева

{

while (root != NULL)

{

Del(root);

}

}

}

char\* String\_for\_anything()

{

char\* tmp = NULL;

int k = 0;

do

{

char s;

s = \_getch();

cout << s;

if (s == 13)

break;

tmp = (char\*)realloc(tmp, sizeof(char) \* (k + 1));

tmp[k] = s;

k++;

} while (true);

tmp = (char\*)realloc(tmp, sizeof(char) \* (k + 1));

tmp[k] = '\0';

cout << endl;

return tmp;

}

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

srand(time(0));

Elem\* a = new Elem, \* b = new Elem, \* c = new Elem;

a->Add("Cat", "Кіт", 3);

b->Add("Car", "Машина", 4);

c->Add("Dog", "Собака", 2);

Tree t;

t.Insert(a);

t.Insert(b);

t.Insert(c);

t.Print(t.GetRoot());

/\*cout << "Введіть слово для пошуку:\n";

char\* key = String\_for\_anything();

Elem\* founded = t.Search(t.GetRoot(), key);

if (founded == NULL)

cout << "Not found" << endl;

else

{

t.Show\_Elem(founded);

}\*/

/\*cout << "Введіть слово для пошуку:\n";

char\* key = String\_for\_anything();

cout << "Введіть новий переклад:\n";

char\* new\_word = String\_for\_anything();

Elem\* founded = t.Redact(key, new\_word);

if (founded == NULL)

cout << "Not found" << endl;

else

{

t.Show\_Elem(founded);

}\*/

/\*cout << "Введіть слово для пошуку:\n";

char\* key = String\_for\_anything();

Elem\* founded = t.Delete\_Translation(key);

if (founded == NULL)

cout << "Not found" << endl;

else

{

t.Show\_Elem(founded);

}\*/

/\*cout << "Введіть слово для пошуку:\n";

char\* key = String\_for\_anything();

cout << "Введіть новий переклад:\n";

char\* new\_word = String\_for\_anything();

Elem\* founded = t.Add\_Translation(key, new\_word);

if (founded == NULL)

cout << "Not found" << endl;

else

{

t.Show\_Elem(founded);

}\*/

t.Show\_Top3\_Popular();

t.Show\_Top\_Least();

return 0;

}