**Министерство науки и высшего образования Российской Федерации**

**федеральное государственное автономное образовательное учреждение высшего образования**

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ**

**УНИВЕРСИТЕТ ИТМО»**

**ФАКУЛЬТЕТ СРЕДНЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ**

**ОТЧЕТ**

**по Лабораторной работе № 19**

Специальность 09.02.07«Информационные системы и программирование»

Дисциплина «Основы алгоритмов и программирование»

|  |  |
| --- | --- |
| Преподаватель:  Яковлева Н.М.  «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_202\_\_г.  Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Выполнил:  студент группы Y2231  Петров Н.В. |

Санкт-Петербург

2019/2020

**Код:**

1. main.cpp

#include <iostream>  
#include "b\_tree.h"  
  
using namespace std;  
  
int main() {  
 int b[] = {1, 2, 3, 4, 5};  
 Node \*root = first(b[1]);  
 for (int i : b)  
 search\_insert(root, i);  
 print\_tree(root, 0);  
 delete\_it(root, search\_insert(root, 2));  
 delete\_it(root, search\_insert(root, 3));  
 cout << endl;  
 print\_tree(root, 0);  
 cout << endl;  
 return 0;  
}

b\_tree.h

#ifndef LAB19V3\_B\_TREE\_H  
#define LAB19V3\_B\_TREE\_H  
  
struct Node {  
 int d;  
 Node \*left;  
 Node \*right;  
};  
  
Node \*first(int d);  
Node \*search\_insert(Node \*root, int d);  
void print\_tree(Node \*p, int level);  
void print\_sorted\_tree(Node \*p);  
void delete\_it(Node \*root, Node \*p);  
  
#endif //LAB19V3\_B\_TREE\_H

b\_tree.cpp

#include <iostream>  
#include "b\_tree.h"  
  
using namespace std;  
  
Node \*first(int d) {  
 Node \*pv = new Node;  
 pv->d = d;  
 pv->left = nullptr;  
 pv->right = nullptr;  
 return pv;  
}  
  
Node \*search\_insert(Node \*root, int d) {  
 Node \*pv = root;  
 Node \*prev;  
 bool found = false;  
 while (pv && !found) {  
 prev = pv;  
 if (d == pv->d) found = true;  
 else if (d < pv->d) pv = pv->left;  
 else pv = pv->right;  
 }  
 if (found) return pv;  
 Node \*pnew = new Node;  
 pnew->d = d;  
 pnew->left = nullptr;  
 pnew->right = nullptr;  
 if (d < prev->d)  
 prev->left = pnew;  
 else  
 prev->right = pnew;  
 return pnew;  
}  
  
void print\_tree(Node \*p, int level) {  
 if (p) {  
 print\_tree(p->left, level + 1);  
 for (int i = 0; i < level; i++) cout << " ";  
 cout << p->d << endl;  
 print\_tree(p->right, level + 1);  
 }  
}  
  
void print\_sorted\_tree(Node \*p) {  
 if (p != nullptr) {  
 print\_sorted\_tree(p->left);  
 cout << p->d << endl;  
 print\_sorted\_tree(p->right);  
 }  
}  
  
Node \*find\_root(Node \*root, Node \*p) {  
 if (root == p) return p;  
 while ((root->right != p) && root->left != p) {  
 if ((p)->d < root->d) root = root->left;  
 else root = root->right;  
 }  
 return root;  
}  
  
Node \*is\_list(Node \*p) {  
 if (p->left != nullptr) {  
 if ((p->left->left == nullptr) and (p->left->right == nullptr)) {  
 return p->left;  
 } else {  
 return nullptr;  
 }  
 } else if (p->right != nullptr) {  
 if ((p->right->left == nullptr) and (p->right->right == nullptr)) {  
 return p->right;  
 } else {  
 return nullptr;  
 }  
 }  
 return nullptr;  
}  
  
void delete\_it(Node \*root, Node \*p) {  
 if (p == nullptr) {  
 cout << "Нельзя удалить несуществующий элемент!" << endl;  
 exit(-1);  
 }  
 Node \*p\_root = find\_root(root, p);  
 if (p->left == nullptr and p->right == nullptr) {  
 if (p\_root->left == p) {  
 p\_root->left = nullptr;  
 }  
 if (p\_root->right == p) {  
 p\_root->right = nullptr;  
 }  
 } else if (p->left == nullptr or p->right == nullptr) {  
 if (p->left == nullptr) {  
 if (p\_root->left == p) {  
 p\_root->left = p->right;  
 } else {  
 p\_root->right = p->right;  
 }  
 } else {  
 if (p\_root->left == p) {  
 p\_root->left = p->left;  
 } else {  
 p\_root->right = p->left;  
 }  
 }  
 } else {  
 Node \*next = p;  
 while (is\_list(next) == nullptr) {  
 if (next->left != nullptr) {  
 next = next->left;  
 } else {  
 next = next->right;  
 }  
 }  
 Node \*tmp\_p = is\_list(next);  
 p->d = tmp\_p->d;  
 if (next->left == tmp\_p) {  
 next->left = nullptr;  
 } else {  
 next->right = nullptr;  
 }  
 }

1. main.cpp

#include <iostream>  
#include "b\_tree.h"  
  
using namespace std;  
  
int main() {  
 int mode;  
 Node \*root = nullptr;  
 Node \*root2 = nullptr;

while (true) {  
 cout << "Выберете режим работы:\n"  
 "1. Новая пара слов\n"  
 "2. Перевести слово\n"  
 "3. Представление словаря по к-ву запросов\n"  
 "4. Вывод словаря\n"  
 "5. Выход из программы\n";  
 cin >> mode;  
 if (mode == 5) return 0;  
 switch (mode) {  
 case 1: {  
 char tmp1[256], tmp2[256];  
 cout << "Введите английское слово: ";  
 cin >> tmp1;  
 cout << "Введите русский перевод: ";  
 cin >> tmp2;  
 if (root != nullptr)  
 search\_insert(root, tmp1, tmp2, false);  
 else  
 root = first(tmp1, tmp2);  
 }  
 break;  
 case 2:  
 char word[256];  
 cout << "Введите английское слово: ";  
 cin >> word;  
 cout << get\_world(root, word) << endl;  
 break;  
 case 3: {  
 delete\_tree(root2);  
 root2 = nullptr;  
 root2 = rebuild\_tree(root, root2);  
 print\_tree(root2, 1);  
 break;  
 }  
 case 4:  
 print\_tree(root, 1);  
 cout << "-----------" << endl;  
 print\_tree(root2, 1);  
 break;  
 default:  
 cout << "Это не число от 1 до 5!" << endl;  
 }  
 }  
}

b\_tree.h

#ifndef LAB19V5\_B\_TREE\_H  
#define LAB19V5\_B\_TREE\_H  
  
struct Node {  
 char \*rus;  
 char \*eng;  
 int counter;  
 Node \*left;  
 Node \*right;  
};  
  
char \*get\_world(Node \*root, char \*word);  
Node \*first(char \*eng, char \*rus);  
Node \*search\_insert(Node \*root, char \*eng, char \*rus = nullptr, int counter = 0, bool sort\_by\_counter = false);  
void print\_tree(Node \*p, int level);  
void print\_sorted\_tree(Node \*p);  
void delete\_it(Node \*root, Node \*p);  
Node \*rebuild\_tree(Node \*p, Node \*s);  
void delete\_tree(Node \*p);  
  
#endif //LAB19V5\_B\_TREE\_H

b\_tree.cpp

#include <iostream>  
#include <cstring>  
#include "b\_tree.h"  
  
using namespace std;  
  
inline void print\_word(char \*rus, char \*eng, int counter) {  
 cout << "ru: " << rus << " en: " << eng << " counter: " << counter;  
}  
  
char \*get\_world(Node \*root, char \*word) {  
 Node \*tmp = search\_insert(root, word);  
 tmp->counter++;  
 return tmp->rus;  
}  
  
Node \*first(char \*eng, char \*rus) {  
 Node \*pv = new Node;  
 pv->eng = new char[strlen(eng) + 1];  
 pv->rus = new char[strlen(rus) + 1];  
 strcpy(pv->eng, eng);  
 strcpy(pv->rus, rus);  
 pv->counter = 0;  
 pv->left = nullptr;  
 pv->right = nullptr;  
 return pv;  
}  
  
Node \*search\_insert(Node \*root, char \*eng, char \*rus, int counter, bool sort\_by\_counter) {  
 Node \*pv = root;  
 Node \*prev;  
 bool found = false;  
 while (pv && !found) {  
 prev = pv;  
 if (!sort\_by\_counter) {  
 if (strcmp(eng, pv->eng) == 0) found = true;  
 else if (strcmp(eng, pv->eng) < 0) pv = pv->left;  
 else pv = pv->right;  
 } else {  
 if (counter == pv->counter) found = true;  
 else if (counter < pv->counter) pv = pv->left;  
 else pv = pv->right;  
 }  
 }  
 if (found) return pv;  
 Node \*pnew = new Node;  
 pnew->eng = new char[strlen(eng) + 1];  
 pnew->rus = new char[strlen(rus) + 1];  
 strcpy(pnew->eng, eng);  
 strcpy(pnew->rus, rus);  
 pnew->counter = counter;  
 pnew->left = nullptr;  
 pnew->right = nullptr;  
 if (!sort\_by\_counter) {  
 if (strcmp(eng, prev->eng) < 0) prev->left = pnew;  
 else prev->right = pnew;  
 } else {  
 if (counter < prev->counter) prev->left = pnew;  
 else prev->right = pnew;  
 }  
 return pnew;  
}  
  
void print\_tree(Node \*p, int level) {  
 if (p) {  
 print\_tree(p->left, level + 1);  
 for (int i = 0; i < level; i++) cout << " ";  
 print\_word(p->rus, p->eng, p->counter);  
 cout << endl;  
 print\_tree(p->right, level + 1);  
 }  
}  
  
void print\_sorted\_tree(Node \*p) {  
 if (p != nullptr) {  
 print\_sorted\_tree(p->left);  
 print\_word(p->rus, p->eng, p->counter);  
 cout << endl;  
 print\_sorted\_tree(p->right);  
 }  
}  
  
Node \*rebuild\_tree(Node \*p, Node \*s) {  
 if (p != nullptr) {  
 if (s == nullptr)  
 s = first(p->rus, p->eng);  
 else  
 search\_insert(s, p->eng, p->rus, p->counter, true);  
 if (p->left != nullptr) rebuild\_tree(p->left, s);  
 if (p->right != nullptr) rebuild\_tree(p->right, s);  
 }  
 return s;  
}  
  
void delete\_tree(Node \*p) {  
 if (p == nullptr) return;  
 if (p->right != nullptr) delete\_tree(p->right);  
 if (p->left != nullptr) delete\_tree(p->left);  
 delete p;  
}  
  
Node \*find\_root(Node \*root, Node \*p) {  
 if (root == p) return p;  
 if (root == nullptr) return nullptr;  
 if (root->left == p or root->right == p) return root;  
 Node \*tmp = find\_root(root->left, p);  
 if (tmp == nullptr) return find\_root(root->right, p);  
 else return find\_root(root->left, p);  
}  
  
Node \*is\_list(Node \*p) {  
 if (p->left != nullptr) {  
 if ((p->left->left == nullptr) and (p->left->right == nullptr)) {  
 return p->left;  
 } else {  
 return nullptr;  
 }  
 } else if (p->right != nullptr) {  
 if ((p->right->left == nullptr) and (p->right->right == nullptr)) {  
 return p->right;  
 } else {  
 return nullptr;  
 }  
 }  
 return nullptr;  
}  
  
void delete\_it(Node \*root, Node \*p) {  
 if (p == nullptr) {  
 cout << "Нельзя удалить несуществующий элемент!" << endl;  
 exit(-1);  
 }  
 Node \*p\_root = find\_root(root, p);  
 if (p->left == nullptr and p->right == nullptr) {  
 if (p\_root->left == p) {  
 delete p\_root->left->rus;  
 delete p\_root->left->eng;  
 delete p\_root->left;  
 p\_root->left = nullptr;  
 }  
 if (p\_root->right == p) {  
 delete p\_root->right->rus;  
 delete p\_root->right->eng;  
 delete p\_root->right;  
 p\_root->right = nullptr;  
 }  
 } else if (p->left == nullptr or p->right == nullptr) {  
 if (p->left == nullptr) {  
 if (p\_root->left == p) {  
 delete p\_root->left->rus;  
 delete p\_root->left->eng;  
 delete p\_root->left;  
 p\_root->left = p->right;  
 } else {  
 delete p\_root->right->rus;  
 delete p\_root->right->eng;  
 delete p\_root->right;  
 p\_root->right = p->right;  
 }  
 } else {  
 if (p\_root->left == p) {  
 delete p\_root->left->rus;  
 delete p\_root->left->eng;  
 delete p\_root->left;  
 p\_root->left = p->left;  
 } else {  
 delete p\_root->right->rus;  
 delete p\_root->right->eng;  
 delete p\_root->right;  
 p\_root->right = p->left;  
 }  
 }  
 } else {  
 Node \*next = p;  
 while (is\_list(next) == nullptr) {  
 if (next->left != nullptr) {  
 next = next->left;  
 } else {  
 next = next->right;  
 }  
 }  
 Node \*tmp\_p = is\_list(next);  
 p->eng = tmp\_p->eng;  
 p->rus = tmp\_p->rus;  
 p->counter = tmp\_p->counter;  
 if (next->left == tmp\_p) {  
 delete p\_root->left->rus;  
 delete p\_root->left->eng;  
 delete next->left;  
 next->left = nullptr;  
 } else {  
 delete p\_root->right->rus;  
 delete p\_root->right->eng;  
 delete next->right;  
 next->right = nullptr;  
 }  
 }  
}

**Результаты и вывод:**

Вывод программы

1.

1  
2  
 3  
 4  
 5  
  
1  
 4  
 5

2.

Выберете режим работы:  
1. Новая пара слов  
2. Перевести слово  
3. Представление словаря по к-ву запросов  
4. Вывод словаря  
5. Выход из программы  
1  
Введите английское слово: Car  
Введите русский перевод: Машина  
Выберете режим работы:  
1. Новая пара слов  
2. Перевести слово  
3. Представление словаря по к-ву запросов  
4. Вывод словаря  
5. Выход из программы  
1  
Введите английское слово: Year  
Введите русский перевод: Год  
Выберете режим работы:  
1. Новая пара слов  
2. Перевести слово  
3. Представление словаря по к-ву запросов  
4. Вывод словаря  
5. Выход из программы  
1  
Введите английское слово: Duck  
Введите русский перевод: Утка  
Выберете режим работы:  
1. Новая пара слов  
2. Перевести слово  
3. Представление словаря по к-ву запросов  
4. Вывод словаря  
5. Выход из программы  
4  
 ru: Машина en: Car counter: 0  
 ru: Утка en: Duck counter: 0  
 ru: Год en: Year counter: 0  
-----------  
Выберете режим работы:  
1. Новая пара слов  
2. Перевести слово  
3. Представление словаря по к-ву запросов  
4. Вывод словаря  
5. Выход из программы  
2  
Введите английское слово: Year  
Год  
Выберете режим работы:  
1. Новая пара слов  
2. Перевести слово  
3. Представление словаря по к-ву запросов  
4. Вывод словаря  
5. Выход из программы  
2  
Введите английское слово: Year  
Год  
Выберете режим работы:  
1. Новая пара слов  
2. Перевести слово  
3. Представление словаря по к-ву запросов  
4. Вывод словаря  
5. Выход из программы  
2  
Введите английское слово: Duck  
Утка  
Выберете режим работы:  
1. Новая пара слов  
2. Перевести слово  
3. Представление словаря по к-ву запросов  
4. Вывод словаря  
5. Выход из программы  
3  
 ru: Машина en: Car counter: 0  
 ru: Утка en: Duck counter: 1  
 ru: Год en: Year counter: 2  
Выберете режим работы:  
1. Новая пара слов  
2. Перевести слово  
3. Представление словаря по к-ву запросов  
4. Вывод словаря  
5. Выход из программы  
4  
 ru: Машина en: Car counter: 0  
 ru: Утка en: Duck counter: 1  
 ru: Год en: Year counter: 2  
-----------  
 ru: Машина en: Car counter: 0  
 ru: Утка en: Duck counter: 1  
 ru: Год en: Year counter: 2  
Выберете режим работы:  
1. Новая пара слов  
2. Перевести слово  
3. Представление словаря по к-ву запросов  
4. Вывод словаря  
5. Выход из программы  
5