МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«Вятский государственный университет»

Факультет автоматики и вычислительной техники Кафедра электронных вычислительных машин

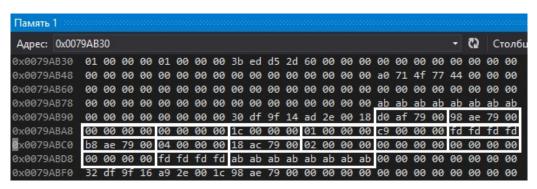
Лабораторная работа № 2 по курсу «Технологии программирования»

Выполнил студент группы ИВТ-21	/Рзаев А. Э./
Проверил доцент кафедры ЭВМ	/Долженкова М. Л./

1 Задание

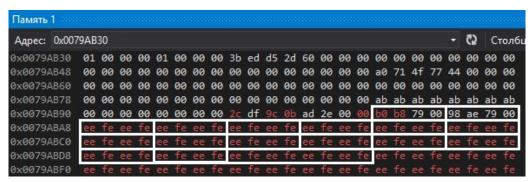
Написать программу для работы с динамической структурой данных (двоичное дерево поиска), содержащей строку и массив чисел с плавающей запятой.

- 2 Работа программы
- 2.1 Занесение элемента



- Указатель на предыдущий занятый элемент
- Указатель на следующий занятый элемент
- Указатель на имя файла подкачки
- Номер строки в файле подкачки
- Размер выделенного участка памяти
- Тип участка памяти
- Количество обращений
- Индикатор начала участка памяти
- Указатель на строку
- Длина строки
- Указатель на массив чисел
- Длина массива
- Указатель на левый дочерний узел
- Указатель на правый дочерний узел
- Указатель на родительский узел
- Индикаторы конца участка памяти

2.2 Удаление элемента



Произошло освобождение области памяти после удаления элемента из дерева.

3 Листинг программы

Листинг разработанной программы приведен в приложении А.

4 Вывод

В ходе выполнения лабораторной работы была разработана структура данных на динамической памяти — двоичное дерево поиска. Изучена структура элемента памяти кучи.

Приложение А (обязательное)

Листинг программы

```
#include <bits\stdc++.h>
struct node
     char* str;
     size_t str_size;
     float* farr;
     size_t farr_size;
     node* left;
     node* right;
     node* parent;
};
struct tree {
     node root; // real root of tree is root.left
};
size t leaks counter = 0;
node* allocate node(size t str size, size t farr size) {
     ++leaks counter;
      node* ret = (node*) realloc(nullptr, sizeof(node));
      ret->str = (char*)realloc(nullptr, sizeof(char) * str size);
      ret->farr = (float*)realloc(nullptr, sizeof(float) * farr size);
      ret->str size = str size;
      ret->farr size = farr size;
      ret->left = ret->right = ret->parent = nullptr;
      std::fill n(ret->str, str size, '\0');
      std::fill n(ret->farr, farr size, 0.0f);
      return ret;
}
node* allocate_node(node* src) {
     node* ret = allocate node(src->str size, src->farr size);
      std::copy(src->str, src->str + src->str size, ret->str);
      std::copy(src->farr, src->farr + src->farr size, ret->farr);
     return ret;
void deallocate_node(node* n) {
      --leaks counter;
     realloc(n->str, 0);
     realloc(n->farr, 0);
     realloc(n, 0);
bool less node(node* first, node* second) {
      if (first->str size == second->str size &&
            std::equal(first->str, first->str + first->str size, second->str))
{ // if first->str == second->str
            return std::lexicographical compare(
                  first->farr, first->farr + first->farr size,
```

```
second->farr, second->farr + second->farr size
                  );
      else {
            return std::lexicographical compare(
                  first->str, first->str + first->str size,
                  second->str, second->str + second->str size
                  );
      }
}
node* find node(tree* tr, node* key) {
      node* root = tr->root.left;
      while (root != nullptr) {
            if (less node(key, root)) { // key < root</pre>
                  root = root->left;
            else if (less node(root, key)) { // key > root
                 root = root->right;
            }
            else { // key == root, success!
                  return root;
      return root; // key not found
void remove node(tree* tr, node* key) {
      node* c = find node(tr, key);
      if (c == nullptr) \{ // no such key in tree, nothing to do
            return;
      }
      node* l = c->left,
            *r = c->right,
            *p = c->parent;
      bool key_is_less = p->left == c;
      if (l == nullptr && r == nullptr) {
            if (key is less) { // left child
                 p->left = nullptr;
            else { //right child
                 p->right = nullptr;
      else if (l == nullptr && r != nullptr) {
            if (key_is_less) {
                 p->left = r;
            else {
                 p->right = r;
      else if (l != nullptr && r == nullptr) {
            if (key is less) {
                 p->left = 1;
            else {
                 p->right = 1;
      else {
```

```
node* new p = r;
            while (r->left != nullptr) {
                  new p = r->left;
            }
            1->parent = new p;
            new_p->left = 1;
            if (key_is_less) {
                  p->left = r;
            }
            else {
                  p->right = r;
            }
      deallocate node(c);
void insert node(tree* tr, node* key) {
      bool key_is_less = true;
      node* root = tr->root.left;
      node* parent = &(tr->root);
      while (root != nullptr) {
            if (less_node(key, root)) { // key < root</pre>
                  key is less = true;
                  parent = root;
                  root = root->left;
            else if (less node(root, key)) { // key > root
                  key is \overline{less} = false;
                  parent = root;
                  root = root->right;
            else { // key == root, node exists, nothing to do
                  return;
      // insert new node
      root = allocate node(key);
      root->parent = parent;
      if (key is less) {
           parent->left = root;
      else {
           parent->right = root;
void deallocate_tree(node* root) {
      if (root == nullptr) {
            return;
      deallocate_tree(root->left);
      deallocate_tree(root->right);
      root->left = root->right = nullptr;
      deallocate_node(root);
}
void allocate tree(tree* tr) {
     tr->root.left = tr->root.right = tr->root.parent = nullptr;
}
void loop(tree* tr) {
      while (true) {
            std::cout << "1. Insert\n2. Find\n3. Remove\n0. Exit\n";</pre>
            int code; std::cin >> code;
```

```
if (code == 1) {
                  std::string s; int n1, n2;
                  std::cin >> n1 >> s >> n2;
                  s.resize(n1);
                  node* key = allocate node(n1 + 1, n2);
                  for (int i = 0; i < n1; ++i) {
                        key->str[i] = s[i];
                  }
                  for (int i = 0; i < n2; ++i) {
                       std::cin >> key->farr[i];
                  insert node(tr, key);
                  deallocate node(key);
            else if (code == 2) {
                  std::string s; int n1, n2;
                  std::cin >> n1 >> s >> n2;
                  s.resize(n1);
                  node* key = allocate node(n1 + 1, n2);
                  for (int i = 0; i < n1; ++i) {
                        key->str[i] = s[i];
                  }
                  for (int i = 0; i < n2; ++i) {
                        std::cin >> key->farr[i];
                  if (find node(tr, key) != nullptr) {
                        std::cout << "key exists\n";</pre>
                  }
                  else {
                        std::cout << "key doesn't exist\n";</pre>
                  deallocate node(key);
            else if (code == 3) {
                  std::string s; int n1, n2;
                  std::cin >> n1 >> s >> n2;
                  s.resize(n1);
                  node* key = allocate node(n1 + 1, n2);
                  for (int i = 0; i < n1; ++i) {
                        key->str[i] = s[i];
                  for (int i = 0; i < n2; ++i) {
                       std::cin >> key->farr[i];
                  remove node(tr, key);
                  deallocate_node(key);
            else if (code == 0) {
                 break;
     }
int main(void) {
     tree tr = { nullptr };
     loop(&tr);
     deallocate tree(tr.root.left);
     std::cout << "leaks: " << leaks counter << std::endl;</pre>
     return 0;
}
```