ГУАП

КАФЕДРА № 34

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Старший преподаватель |  |  |  | Жиданов К.А. |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ |
|  |
| по курсу: |
| ЯЗЫКИ ПРОГРАММИРОВАНИЯ |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. | 3145 |  |  |  | Калинин Е.Д. |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2022

**Вариант №4**

Дерево (добавление, удаление)

**Цель работы**

Реализовать АТД (абстрактный тип данных) в виде пользовательского типа данных и

набора функций, реализующих заданные операции. Помимо стандарных интерфейсов (чтение/добавление/поиск/удаление), требуется реализовать чтение/выгрузку данных

из файла.

**Ход работы**

Составим программу дерева:

#include "tree.h"

#include <stdio.h>

#include <stdlib.h>

int unit\_test() {

treenode \*n0, \*n11, \*n12, \*n13, \*n2;

n0 = add\_node(NULL, 5);

if (n0 == NULL)

return -1;

n11 = add\_node(n0, 10);

if (n11 == NULL)

return -1;

n12 = add\_node(n0, 23);

if (n12 == NULL)

return -1;

n13 = add\_node(n0, 8);

if (n13 == NULL)

return -1;

n2 = add\_node(n13, 11);

if (n2 == NULL)

return -1;

return 0;

}

int main(int argc, char\*\* argv) {

int result = unit\_test();

tree\_from\_file("values.txt"); //результат чтения файла вида (<Tab><value>\n)

return result; //результат юнит-теста

}

/\*

ГРАФИЧЕСКОЕ ПРЕДСТАВЛЕНИЕ ТЕСТОВОГО ДЕРВА:

n0

/ | \

n11 n12 n13

\

n2

\*/

define \_CRT\_SECURE\_NO\_WARNINGS

#include "tree.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define BUFFER\_SIZE 10

descendants\* add\_descendant(descendants\* list, treenode\* node) {

if (list != NULL) { //добавление в список потомков

descendants\* new = (descendants\*)malloc(sizeof(descendants));

if (new == NULL)

return NULL;

if (list->value == NULL) {

list->value = node;

list->next = NULL;

return list;

}

while (list->next != NULL)

list = list->next;

list->next = new;

new->value = node;

new->next = NULL;

return new;

}

else { //создание списка потомков

descendants\* new = (descendants\*)malloc(sizeof(descendants));

if (new == NULL)

return NULL;

new->value = NULL;

new->next = NULL;

return new;

}

}

treenode\* add\_node(treenode\* node, int value) {

if (node != NULL) { //добавление к существующему узлу

treenode\* leaf = (treenode\*)malloc(sizeof(treenode));

if (leaf == NULL)

return NULL;

leaf->value = value;

leaf->ancestor = node;

leaf->descendant = add\_descendant(NULL, NULL);

add\_descendant(node->descendant, leaf);

return leaf;

}

else { //создание корня

treenode\* root = (treenode\*)malloc(sizeof(treenode));

if (root == NULL)

return NULL;

root->value = value;

root->ancestor = NULL;

root->descendant = add\_descendant(NULL, NULL);

return root;

}

}

void show\_descendants(treenode\* node) { //процедура вывода листьев узла

descendants\* list = node->descendant;

if (list->value == NULL) {

printf("There are no descenants of node %d", node->value);

return;

}

treenode\* leaf;

while (list->next != NULL) {

leaf = list->value;

printf("%d\n", leaf->value);

list = list->next;

}

leaf = list->value;

printf("%d\n", leaf->value);

return;

}

void tree\_from\_file(const char\* filename) {

FILE\* f = fopen(filename, "r");

char buffer[BUFFER\_SIZE];

char temp[4];

unsigned int i, j, count, level = 0;

int value;

treenode\* root = NULL;

treenode\* current = NULL;

treenode\* previous = NULL;

while (!feof(f)) {

while (fgets(buffer, BUFFER\_SIZE, f) != NULL) {

j = 0;

count = 0;

memset(temp, 0, 4 \* sizeof(char));

for (i = 0; i < BUFFER\_SIZE; i++) {

if (buffer[i] != ' ') {

temp[j] = buffer[i];

j++;

}

if (buffer[i] == ' ')

count++;

}

value = atoi(temp);

if (count > 0) {

if (count > level) {

previous = current;

current = add\_node(current, value);

level = count;

}

else

if (count < level) {

current = previous->ancestor;

previous = current;

current = add\_node(current, value);

level = count;

}

else {

current = previous;

previous = current;

current = add\_node(current, value);

level = count;

}

}

else {

root = add\_node(NULL, value);

current = root;

}

}

}

///

printf("Descendants of root \"156\":\n");

show\_descendants(root); //должно выдать 4, 3, 13

printf("Descendants of node \"4\":\n");

show\_descendants(((descendants\*)root->descendant)->value); //должно выдать 2

///

}

#pragma once

#ifndef \_\_TREE\_\_

#define \_\_TREE\_\_

typedef struct {

struct treenode\* value;

struct descendants\* next;

} descendants; //структура списка

typedef struct {

int value;

struct treenode\* ancestor;

struct descendants\* descendant;

} treenode; //структура дерева

descendants\* create\_descendant();

descendants\* add\_descendant(descendants\* list, treenode\* node);

treenode\* create\_tree(int value);

treenode\* add\_node(treenode\* node, int value);

void show\_descendants(treenode\* node);

void tree\_from\_file(const char\* filename);

#endif

При запуске программа выполняется успешно.