**Отчет по лабораторной работе №** 23по курсу “Фундаментальная информатика”

Студент группы М80-101Б-21 Ершова Станислава Григорьевича, № по списку 8

Контакты e-mail, telegram: [stas.ershov57@gmail.com](mailto:stas.ershov57@gmail.com) , @stas\_orel

Работа выполнена: «24» марта 2022г.

Преподаватель: каф. 806 Севастьянов Виктор Сергеевич

Отчет сдан «24» марта 2021 г., итоговая оценка \_\_\_\_\_\_\_\_

Подпись преподавателя \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* 1. **Тема:** **Динамические структуры данных. Обработка деревьев.**

1. **Цель работы: Научиться работать с динамическими структурами данных и деревьями.**
2. **Задание** (*вариант №* **22**)**: Определить число вершин дерева.**
3. **Оборудование** (студента):

Процессор *Intel Pentium N4200 1.1 ГГц* с ОП *8* Гб, SSD *128* Гб. Монитор *1920x1080*

1. **Программное обеспечение (**студента**):**

Операционная система семейства: *linux*, наименование: *ubuntu*, версия *20.04.3 LTS*

интерпретатор команд: *bash* версия *5.0.17*

Система программирования -- версия --**,** редактор текстов *nano* версия *4.8*

Утилиты операционной системы --

Прикладные системы и программы --

Местонахождение и имена файлов программ и данных на домашнем компьютере --

**6. Идея, метод, алгоритм:**

**7. Сценарий выполнения работы** [план работы,первоначальный текст программы в черновике(можно на отдельном листе)итесты либо соображения по тестированию].

**8. Распечатка протокола** (подклеить листинг окончательного варианта программы с тестовыми примерами,подписанныйпреподавателем).

#include <stdio.h>

#include <malloc.h>

#include <string.h>

#include <stdbool.h>

typedef struct \_node {

struct \_node\* parent;

int value;

int current\_cnt;

int available\_cnt;

struct \_node\*\* children;

} node;

int STANDART\_ARRAY\_SIZE = 5;

int vertex\_cnt = 0;

node\* genesis\_node;

bool\* borders;

int max(int a, int b) {

if (a > b)

return a;

return b;

}

void create\_node(node\* u, int v) {

node\* nd = malloc(sizeof(node));

nd->parent = u;

nd->value = v;

nd->current\_cnt = 0;

nd->available\_cnt = STANDART\_ARRAY\_SIZE;

nd->children = malloc(sizeof(node) \* STANDART\_ARRAY\_SIZE);

if(u->current\_cnt == u->available\_cnt - 1) {

u->children = realloc(u->children, sizeof(node) \* u->available\_cnt \* 2);

u->available\_cnt \*= 2;

}

u->children[u->current\_cnt] = nd;

u->current\_cnt++;

}

node\* create\_genesis\_node(int v) {

node\* nd = malloc(sizeof(node));

nd->parent = NULL;

nd->value = v;

nd->current\_cnt = 0;

nd->available\_cnt = STANDART\_ARRAY\_SIZE;

nd->children = malloc(sizeof(node) \* STANDART\_ARRAY\_SIZE);

return nd;

}

void delete\_node(node\* v) {

for (int i = v->current\_cnt - 1; i >= 0; --i) {

delete\_node(v->children[i]);

}

node\* v\_parent = v->parent;

int del\_index = 0;

for (int i = 0; i < v\_parent->current\_cnt; ++i) {

if (v\_parent->children[i]->value == v->value) {

del\_index = i;

vertex\_cnt--;

free(v);

break;

}

}

for (int i = del\_index + 1; i < v\_parent->current\_cnt; ++i) {

v\_parent->children[i - 1] = v\_parent->children[i];

}

v\_parent->current\_cnt--;

}

node\* find\_vertex(node\* u, int vertex) {

if (u->value == vertex)

return u;

node\* vertex\_address = NULL;

for (int i = 0; i < u->current\_cnt; ++i) {

vertex\_address = find\_vertex(u->children[i], vertex);

if (vertex\_address != NULL && vertex\_address->value == vertex)

break;

}

return vertex\_address;

}

int str\_to\_int(char\* s) {

int ans = 0;

for (int i = 0; i < strlen(s); ++i) {

if (s[i] >= '0' && s[i] <= '9') {

ans = ans \* 10 + (s[i] - '0');

} else {

return -2022;

}

}

return ans;

}

void print\_tree(node\* u, int space\_cnt) {

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

if (sizeof(borders) / sizeof(bool) - 1 < space\_cnt)

borders = realloc(borders, sizeof(borders) \* 2);

borders[space\_cnt] = true;

for (int i = 0; i < u->current\_cnt; ++i) {

if (i == u->current\_cnt - 1)

borders[space\_cnt] = false;

for (int j = 0; j < space\_cnt; ++j) {

if (borders[j])

printf("|");

else

printf(" ");

}

printf("|\n");

for (int j = 0; j < space\_cnt; ++j) {

if (borders[j])

printf("|");

else

printf(" ");

}

printf("+");

printf("----");

print\_tree(u->children[i], space\_cnt + 5);

}

borders[space\_cnt] = false;

}

void print\_info(bool first\_print) {

if (first\_print) {

printf(" Доступные операции:\n");

printf("0 - Вывести справку\n");

printf("1 - Добавить ребро в дерево\n");

printf("2 - Удалить ребро из дерева\n");

printf("3 - Вывести дерево\n");

printf("4 - Вывести количество вершин в дереве\n");

} else {

printf("Сначала в отдельной строке вводится номер операции 0-4\n");

printf("После чело для операции 1 - 2 вершины,\n");

printf(" для операции 2 - 1 вершина,\n");

printf(" для операций 3 и 4 дополнительные входные данный не требуются.\n");

}

}

int main() {

borders = malloc(sizeof(bool) \* 30);

bool is\_genesis = true;

char str[100];

print\_info(true);

while (true) {

printf("Операция: ");

scanf("%s", str);

int op = str\_to\_int(str);

if (op < 0 || op > 4) {

printf("Введенные данные некорректны\n");

continue;

}

if(op == 0) {

print\_info(false);

} else if (op == 1) {

int u, v;

char u1[100], v1[100];

scanf("%s %s", u1, v1);

u = str\_to\_int(u1);

v = str\_to\_int(v1);

if (u < 0 || v < 0) {

printf("Введены некоректные данные, допустимы только целочесленные неотрицательные значения вершин\n");

continue;

}

if (is\_genesis) {

is\_genesis = false;

genesis\_node = create\_genesis\_node(u);

create\_node(genesis\_node, v);

vertex\_cnt += 2;

} else {

node\* u\_address = find\_vertex(genesis\_node, u);

if (u\_address == NULL) {

printf("Вершина %d отсутсвует в дереве\n", u);

continue;

}

node\* v\_address = find\_vertex(genesis\_node, v);

if (v\_address == NULL) {

create\_node(u\_address, v);

vertex\_cnt++;

} else {

printf("Такая вершина уже есть в дереве\n");

}

}

} else if (op == 2) {

char u\_str[100];

int u;

scanf("%s", u\_str);

u = str\_to\_int(u\_str);

node\* v = find\_vertex(genesis\_node, u);

if (v == NULL)

printf("Введены некорректные данные\n");

else

delete\_node(v);

} else if (op == 3) {

print\_tree(genesis\_node, 0);

} else if (op == 4) {

printf("Вершин в дереве: %d\n", vertex\_cnt);

}

}

}

**9.Дневник отладки** должен содержать дату и время сеансов отладки и основные события(ошибки в сценарии и программе,нестандартные ситуации) и краткие комментарии к ним. В дневнике отладки приводятся сведения об использовании других ЭВМ, существенном участии преподавателя и других лиц в написании и отладке программы.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Лаб. | Дата | Время | Событие | Действие по исправлению | Примечание |
|  | или |  |  |  |  |  |
|  | дом. |  |  |  |  |  |
|  |  |  |  |  |  |  |

1. **Замечания автора** по существу работы
2. **Выводы: Разобрался как работает динамическая память, надеюсь в будущем буду по минимуму с этим связываться, не понравилось. Научился делать крутой вывод дерева.**

Подпись студента \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_