**Отчет по лабораторной работе № 23** по курсу “Языки и методы программирования”

Студент группы М80-103Б-21 Фадеев Денис Вадимович, № по списку 22

Контакты e-mail: denfad2003@mail.ru, telegram: @Denissimo\_f

Работа выполнена: «26» марта 2022г.

Преподаватель: каф. 806 Севастьянов Виктор Сергеевич

Отчет сдан « » \_\_\_\_\_\_\_\_\_20\_\_ г., итоговая оценка \_\_\_\_\_\_\_\_

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

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

1. **Цель работы:** Составить программу на языке Си для построения и обработки дерева общего вида, содержащего узлы типа int.
2. **Задание:** вариант №5 Определить значение нетерминальной вершины дерева с максимальной глубиной.
3. **Оборудование** :

Процессор *Intel Core i5-6200U @ 4x 2.30GHz* с ОП 16 Гб, НМД 512 Гб. Монитор *1920x1080*

1. **Программное обеспечение :**

Операционная система семейства: *linux*, наименование: *ubuntu*, версия 20*.04 focal*

интерпретатор команд: *bash* версия 5.0.16(1)-release.

Редактор текстов *emacs* версия 26.3

**6. Идея, метод, алгоритм**

Необходимо написать отдельный файл с Си кодом, в котором будут реализованы основные методы для работы с деревом, а также разработать необходимый тип данных Unit, выполняющий роль узлов дерева.

**7.Сценарий выполнения работы**

1) чтобы реализовать создание дерева, напишем функцию, выделяющую участок памяти размером с тип данных Unit, под наше дерево и возвращающую ссылку на него.

2) функция добавления элемента в дерево ищет по ключу будущий родительский узел, расширяет выделяемую в нём память под сыновей и вставляет туда новый узел.

3) чтобы удалить элемент дерева, функция removeUnit подчищает все дочерние узлы элемента, просто очищая память от них. Затем удаляется из памяти и сам заданный элемент.

4) вывод дерева происходит рекурсивным обходом с учетом глубины входа в дерево. Чем глубже вход, тем дальше от левой границы терминала будет отображаться значение элемента

5) специальная функция поиска нетерминального элемента с максимальной глубиной выполняется рекурсивно. Она доходит до листьев дерева, и возвращает их, только в переменную count записана их глубина. После чего я нахожу самое глубокое из них, нахожу ссылку на него, а затем и отцовский элемент, который не является терминальным.

Выполнение программы:

**Working with tree. Enter code of command.**

**Code 1: Create tree**

**Code 2: Add unit**

**Code 3: Delete unit**

**Code 4: Print tree**

**Code 5: Special function**

**Code 6: Exit programm**

**Enter:** 1

**Enter key of first unit:**5

**Tree created**

**Working with tree. Enter code of command.**

**Code 1: Create tree**

**Code 2: Add unit**

**Code 3: Delete unit**

**Code 4: Print tree**

**Code 5: Special function**

**Code 6: Exit programm**

**Enter:** 2

**Enter parent key:** 5

**Enter key of unit:** 6

**Unit with key 6 created**

**Working with tree. Enter code of command.**

**Code 1: Create tree**

**Code 2: Add unit**

**Code 3: Delete unit**

**Code 4: Print tree**

**Code 5: Special function**

**Code 6: Exit programm**

**Enter:** 2

**Enter parent key:** 5

**Enter key of unit:** 7

**Unit with key 7 created**

**Working with tree. Enter code of command.**

**Code 1: Create tree**

**Code 2: Add unit**

**Code 3: Delete unit**

**Code 4: Print tree**

**Code 5: Special function**

**Code 6: Exit programm**

**Enter:** 2

**Enter parent key:** 7

**Enter key of unit:** 8

**Unit with key 8 created**

**Working with tree. Enter code of command.**

**Code 1: Create tree**

**Code 2: Add unit**

**Code 3: Delete unit**

**Code 4: Print tree**

**Code 5: Special function**

**Code 6: Exit programm**

**Enter:** 2

**Enter parent key:** 6

**Enter key of unit:** 3

**Unit with key 3 created**

**Working with tree. Enter code of command.**

**Code 1: Create tree**

**Code 2: Add unit**

**Code 3: Delete unit**

**Code 4: Print tree**

**Code 5: Special function**

**Code 6: Exit programm**

**Enter:** 2

**Enter parent key:** 6

**Enter key of unit:** 9

**Unit with key 9 created**

**Working with tree. Enter code of command.**

**Code 1: Create tree**

**Code 2: Add unit**

**Code 3: Delete unit**

**Code 4: Print tree**

**Code 5: Special function**

**Code 6: Exit programm**

**Enter:** 4

**5**

**|>6**

**| |>3**

**| |>9**

**|>7**

**| |>8**

**Working with tree. Enter code of command.**

**Code 1: Create tree**

**Code 2: Add unit**

**Code 3: Delete unit**

**Code 4: Print tree**

**Code 5: Special function**

**Code 6: Exit programm**

**Enter:** 5

**Key of special unit 6**

**Working with tree. Enter code of command.**

**Code 1: Create tree**

**Code 2: Add unit**

**Code 3: Delete unit**

**Code 4: Print tree**

**Code 5: Special function**

**Code 6: Exit programm**

**Enter:** 2

**Enter parent key:** 9

**Enter key of unit:** 1

**Unit with key 1 created**

**Working with tree. Enter code of command.**

**Code 1: Create tree**

**Code 2: Add unit**

**Code 3: Delete unit**

**Code 4: Print tree**

**Code 5: Special function**

**Code 6: Exit programm**

**Enter:** 5

**Key of special unit 9**

**Working with tree. Enter code of command.**

**Code 1: Create tree**

**Code 2: Add unit**

**Code 3: Delete unit**

**Code 4: Print tree**

**Code 5: Special function**

**Code 6: Exit programm**

**Enter:** 6

**End program**

**8. Распечатка протокола**

**tree.c**

#include "tree.h"

unit \* makeTree(int key) {

unit \*u = malloc(sizeof(unit));

u -> key = key;

u -> count = 0;

u -> sons = NULL;

**return u;**

}

unit \* findUnit(unit \* u, int key) {

if(u -> key == key){

return u;

}

if(u -> count > 0){

for(int i = 0; i < u->count; i++) {

unit \*p = findUnit(u->sons[i], key);

if(p!=NULL) return p;

}

return NULL;

} else {

return NULL;

}

}

void addUnit(unit \*u, int parentKey, int key) {

if(findUnit(u, key) != NULL) printf("UNIT EXISTS\n");

else {

unit \*f = findUnit(u, parentKey);

if(f!=NULL) {

if(f->count > 0){

f -> sons = (unit \*\*)realloc(f->sons, sizeof(unit \*) \* (f->count + 1));

f -> sons[f->count] = makeTree(key);

f->count ++;

} else if(f->count == 0){

f->sons = (unit\*\*)malloc(sizeof(unit\*));

f -> sons[0] = makeTree(key);

f->count = 1;

}

}

else {

printf("PARENT NOT FOUND\n");

}

}

}

void clearTree(unit \*u) {

for(int i = 0; i < u -> count; i++) {

if(u -> sons[i] != NULL) {

clearTree(u -> sons[i]);

}

}

free(u -> sons);

u -> sons = NULL;

free(u);

u = NULL;

}

unit\* findParent(unit \*f, unit \*p){

if(f->count>0){

for(int i = 0; i<f->count; i++){

if(f->sons[i]->key == p->key){

return f;

}

else{

unit\* s = findParent(f->sons[i], p);

if(s!=NULL) return s;

}

}

return NULL;

}

**else return NULL;**

}

void removeUnit(unit \*u, unit \*f) {

if(findUnit(u, f->key) == NULL){

printf("UNIT NOT FOUND");

}

else {

unit \*p = findParent(u, f);

int rmv = 0;

for(int i = 0; i<p->count; i++){

if(p->sons[i]->key == f->key){

rmv = i;

break;

}

}

clearTree(f);

for(int i = rmv; i<p->count-1; i++){

p->sons[i] = p->sons[i+1];

}

p->count --;

}

}

void printSons(unit \*u, int deep){

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

printf("| ");

if(deep != 0)

printf("|");

printParent(u, deep);

printf("\n");

for(int i = 0; i < u->count; i++){

printSons(u->sons[i], deep+1);

}

}

void printParent(unit \*u, int deep){

if(deep != 0)

printf(">%d", u->key);

else

printf("%d", u->key);

}

void printTree(unit \*u){

printSons(u, 0);

}

unit \* findMCNTU(unit \*u, int iter) {

unit \*p = NULL;

int max = 0;

if(u -> count < 1 && iter == 0){

return NULL;

}

else if(u ->count < 1){

unit \*s = malloc(sizeof(unit));

s -> key = u -> key;

s -> count = iter;

s -> sons = NULL;

return s;

}

else{

for(int i = 0; i < u->count; i++){

if(findMCNTU(u->sons[i], iter+1)->count > max){

p = findMCNTU(u->sons[i], iter+1);

max = p->count;

}

}

return p;

}

}

unit \* specFunc(unit \* u){

unit \*p = findMCNTU(u, 0);

if(p == NULL){

return NULL;

}

else {

unit \*s = findUnit(u, p->key);

unit \*f = findParent(u, s);

return f;

}

}

**tree.h**

#ifndef tree\_h

#define tree\_h

#include <stdio.h>

#include "stdlib.h"

typedef struct unit {

struct unit \*\* sons;

int key;

int count;

} unit;

unit \* makeTree(int key);

void clearTree(unit \* u);

void printParent (unit \*u, int deep);

void printSons (unit \*u, int deep);

void printTree (unit \*u);

unit \* findUnit (unit \*u, int key);

void addUnit(unit \* u, int parentKey, int key);

unit \*findParent (unit \*f, unit \*p);

void removeUnit(unit \*u, unit \*f);

unit \* findMCTU(unit \*u, int iter);

unit \* specFunc(unit \*u);

#endif /\* tree\_h \*/

**main.c**

#include "tree.h"

#include "stdlib.h"

#include "stdbool.h"

#include "string.h"

int main() {

// insert code here...

unit \*tree = NULL;

bool working = true;

while(working) {

int input = 0;

char h[] = "";

while(input == 0){

printf("Working with tree. Enter code of command.\n\tCode 1: Create tree\n\t\tCode 2: Add unit\n\t\t\tCode 3: Delete unit\n\t\t\t\tCode 4: Print tree\n\t\t\t\t\tCode 5: Special function\n\t\t\t\t\t\tCode 6: Exit programm\nEnter: ");

scanf("%s", h);

if(!strcmp("1",h)){

input = 1;

}

else if(!strcmp("2",h)) {

input = 2;

}

else if(!strcmp("3",h)) {

input = 3;

}

else if(!strcmp("4",h)) {

input = 4;

}else if(!strcmp("5",h)) {

input = 5;

}else if(!strcmp("6",h)) {

input = 6;

}else {

printf("Incomprehensible input\n");

}

}

switch(input){

case 1:

{

if(tree != NULL)

printf("Tree already exists\n");

else {

int key = 0;

printf("Enter key of first unit:");

scanf("%d", &key);

tree = makeTree(key);

printf("Tree created\n");

}

break;

}

case 2:

{

if(tree != NULL){

int key = 0;

int parentKey = 0;

printf("Enter parent key: ");

scanf("%d", &parentKey);

printf("Enter key of unit: ");

scanf("%d", &key);

addUnit(tree, parentKey, key);

printf("Unit with key %d created\n", key);

}

else printf("Tree not exists\n");

break;

}

case 3:

{

if(tree != NULL) {

int key = 0;

printf("Enter key of unit for deliting: ");

scanf("%d", &key);

if(findUnit(tree, key) != NULL) {

unit \*p = findUnit(tree, key);

removeUnit(tree, p);

printf("Unit removed\n");

}

else {

printf("Unit not found\n");

}

}

else {

printf("Tree not exists\n");

}

break;

}

case 4: {

if(tree != NULL) {

printTree(tree);

}

else {

printf("Tree not exists\n");

}

break;

}

case 5:

printf("Key of special unit %d\n", specFunc(tree)->key);

break;

case 6:

printf("End program\n");

working = false;

break;

}

}

return 0;

}

**9. Дневник отладки**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Лаб. Или Дом. | Дата | Время | Событие | Действие по исправлению | Примечание |
|  | - | - | - | - | - | - |

1. **Замечания автора** -
2. **Выводы**

Результатом выполнения лабораторной работы стало глубокое изучение работы с памятью на языке Си, использование деревьев, реализация их на таком низком уровне. Отдельно могу выделить работу с памятью и ссылками. Это достаточно интересное занятие, которое даёт возможность ощутить и прикоснуться на низком уровне к аппаратным средствам компьютера. Реализация дерева оказалось не простой задачей, но если потихоньку выполнять алгоритм и четко осознавать, что происходит на каждом шаге, то ничего сложного в задании нет. На мой взгляд, на данный момент деревья не самый востребованный структур данных, её применимость ограничена.

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