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

Студент группы М8О-103Б-21

Батулин Евгений Андреевич, № по списку 2

Контакты: e-mail: uggin@inbox.ru, telegram: @uggin0

Работа выполнена: «23» апреля 2021 г.

Преподаватель: каф. 806 Севастьянов Виктор Сергеевич

Отчет сдан «  » _____20__ г., итоговая оценка _____

Подпись преподавателя _____

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

2. **Цель работы:** составить программу на языке Си для построения и обработки дерева общего вида или упорядоченного двоичного дерева, содержащего узлы типа float, int, char или enum

3. **Задание:** проверить, находятся ли все листья дерева на одном уровне
4. **Оборудование** (студента):
Процессор *Intel Core i9-9980HK(QQLS), 8c/16t @ 4.4GHz* с ОП *32768* Мб, НМД *6656* Гб. Монитор *1920x1080*

5. **Программное обеспечение (**студента**):**
Операционная система семейства: *Windows*, наименование: *10*, версия *1809 LTSC*
интерпретатор команд: *MSYS* версия *1.3.0.0*.
Система программирования -- версия --, редактор текстов *Visual Studio Code*, версия *1.66.2*
Утилиты операционной системы:
Прикладные системы и программы: gcc

6. **Идея, метод, алгоритм** решения задачи

Каждый узел будет содержать в себе ключ в int, количество детей/младших узлов в int, массив указателей на младшие узлы. Для полноценной работы программы я создаю функции создания узла, добавления узла, поиска узла (циклическая проверка ключей у младших узлов), удаления узла (через поиск узла и освобождение памяти) и вывода сгенерированного дерева, а также основной цикл с выполнением программы и текстовым интерфейсом. Для решения задания по варианту я создал ещё одну функцию – поиск максимальной глубины ветви.

7. **Сценарий выполнения работы**

1. Запуск среды программирования
2. Создание программы
3. Проверка работоспособности программы на различных данных, вводимых человеком
4. Отладка
5. Протоколирование работы отлаженной программы
6. Завершение работы

| **Входные данные** | **Выходные данные** | **Описание тестируемого случая** |
|---|---|---|
| 1 10 1 10 20 1 20 40 1 20 34<br>1 34 45 2 | 10<br>  20<br>  40<br>  34<br>  45 | Проверка работы программы в типичной ситуации |
| 1 10 1 10 20 1 20 40 1 20 34<br>1 34 45 1 20 28<br>3 34 2 | 10<br>  20<br>  40<br>  28 | Попытка удаления ветви целиком, проверка на сохранения порядка у младших узлов |

| | | |
|---|---|---|
| 1 5915<br>1 5915 932<br>14<br>1 5915 540<br>1 932 14<br>14 | 1<br><br><br>0 | Проверка функции по варианту задания |

## 8. Распечатка протокола

main.c:

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

typedef struct _node {
        int key;
   int q;
        struct _node ** sons;
} node;

node *make_node(int x) {
        node *n = malloc(sizeof(node));
        n->key = x;
   n->q = 0;
        n->sons = NULL;
        return n;
}

node * find_node(node *n, int father_key) {
   if (n->key == father_key) {
      return n;
   }
   if (n->q > 0) {
      for (int i = 0; i < n->q; ++i){
         node * son = find_node(n->sons[i], father_key);
         if (son != NULL) {
            return son;
         }
      }
      return NULL;
   } else {
      return NULL;
   }
}


void add_node(node * n, int father_key, int key) {
   if (find_node(n, key) == NULL) {
      node *parent = find_node(n, father_key);
      if (parent == NULL) {
         printf("Can't find specified parent\n");
         exit;
      }
      else {
         if (parent->q != 0) {
            parent->sons = (node **)realloc(parent->sons, sizeof(node *) * (parent->q + 1));
            parent->sons[parent->q] = make_node(key);
            parent->q++;
         } else if (parent->q == 0) {
            parent->sons = (node **)malloc(sizeof(node *));
            parent->sons[parent->q] = make_node(key);
            parent->q++;
         }
      }
   } else {
      printf("Already exists\n");
   }
}

int max_depth (node *n, int level)
{
   if (n->q == 0) {
      return level;
   } else if (n->q > 0) {
      for (int i = 0; i < n->q; i++){
         max_depth(n->sons[i], level + 1);
      }
   } else {
      return -1;
   }
}

int leafs(node *n, int level) {
   printf("key: %d q: %d\n", n->key, n->q);
   if (n->q == 0) {
      return 1;
   } else if (n->q == 1) {
```

```
        leafs(n->sons[0], level);
    } else if (n->q > 0) {
        for (int i = 0; i < n->q - 1; i++){
            int md1 = max_depth(n->sons[i], level);
            int md2 = max_depth(n->sons[i+1], level);
            if (md1 != md2) {
                printf("No");
                return 0;
            }
        }
    } else {
        printf("Yes");
        return 1;
    }
}


void free_node(node * n) {
    for (int i = 0; i < n->q; ++i){
        if (n->sons[i] != NULL) {
                                free_node(n->sons[i]);
        }
    }
    free(n->sons);
    n->sons = NULL;
                free(n);
}

void remove_element(node * n, int key) {
    if (n->key == key) {
        free_node(n);
    } else {
        for (int i = 0; i < n->q; ++i){
        if (n->sons[i]->key == key) {
                                free_node(n->sons[i]);
            for (int k = i; k < n->q - 1; k++) {
                n->sons[k] = n->sons[k+1];
            }
            n->q--;
        } else {
                                remove_element(n->sons[i], key);
        }
        }
    }
}

void print_tree_beauty(node *n, int deep) {
    if (n->key != NULL) {
        for (int i = 0; i < deep; i++) {
            printf("\t");
        }
        printf("%d\n", n->key);
    }
    if (n->q > 0) {
        for (int i = 0; i < n->q; i++){
            print_tree_beauty(n->sons[i], deep + 1);
        }

    }
}

void print_tree(node *n) {
                print_tree_beauty(n, 0);
}

int main() {
    printf("_____________________________________________________\n");
    printf("|                   Tree Editor                     |\n");
    printf("_____________________________________________________\n");
    printf("| Add Element - |1| Show tree - |2| Remove Element - |3| |\n");
    printf("|   Are leafs on the same depth? - |14|   Exit - |0|     |\n");
    printf("_____________________________________________________\n");


    char input[] = "";
    bool execute = true;
    bool isTreeCreated = false;
    int task = -1;
    node *n;

    while (execute) {
        char input[] = "";
        printf("\n");
        scanf("%s", input);

        if (!strcmp("0", input)) task = 0;
        if (!strcmp("1", input)) task = 1;
        if (!strcmp("2", input)) task = 2;
        if (!strcmp("3", input)) task = 3;
        if (!strcmp("14", input)) task = 14;

        switch (task) {
            case 0:
                execute = false;
```

```
            if (isTreeCreated) {
                free_node(n);
            }
            break;
        case 1:
            if (isTreeCreated) {
                int keyp = 0;
                int key = 0;
                char p[] = "";
                char e[] = "";
                printf("Enter parent key\n");
                scanf("%s", p);
                keyp = atoi(p);
                printf("Enter element key\n");
                scanf("%s", e);
                key = atoi(e);
                add_node(n, keyp, key);
            } else {
                int key = 0;
                char e[] = "";
                printf("Creating new tree...\n");
                printf("Enter element key\n");
                scanf("%s", e);
                key = atoi(e);
                n = make_node(key);
                isTreeCreated = true;
            }
            break;
        case 2:
            if (isTreeCreated) {
                print_tree(n);
            } else {
                printf("Binary tree is not created\n");
            }
            break;
        case 3:
            if (isTreeCreated) {
                int key = 0;
                int rootkey = n->key;
                char e[] = "";
                printf("Enter element key\n");
                scanf("%s", e);
                key = atoi(e);
                remove_element(n, key);
                if (key == rootkey) isTreeCreated = false;
            } else {
                printf("Binary tree is not created\n");
            }
            break;
        case 14:
            if (isTreeCreated) {
                printf("%i",leafs(n,0));
            } else {
                printf("Binary tree is not created\n");
            }
            break;
        default:
            printf("Incorrect input");
            break;
    }
  }
          return 0;
}
```

**9. Дневник отладки**

| № | Лаб. или дом. | Дата | Время | Событие | Действие по исправлению | Примечание |
|---|---|---|---|---|---|---|
| 1 | дом | 23\04\2022 | 8:34 | Неадекватная работа функции по варианту | Использование внешней функции для поиск глубины ветви внутри это функции | |

**10. Замечания автора** по существу работы

**11. Выводы**

Подводя итог, в процессе данной работы я укрепил свои навыки работы с динамическими структурами данных, научился практически реализовывать нормальные деревья и несколько улучшил свои знания о языке Си.

Подпись студента _____