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

Студент группы М80-103Б-21 Тысячный Владислав Валерьевич, № 21 по списку

Контакты e-mail: tysycny2003@gmail.com,
telegram: @Bradvurt

Работа выполнена: «26» марта 2021г.

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 упорядоченного бинарного дерева, содержащего узлы типа int.

3. **Задание №16:** Проверить, является ли двоичное дерево симметричным

4. **Оборудование** :

   Процессор *IIntel® Core™ i7-7700HQ CPU @ 2.80GHz × 8* с ОП *7,7* Гб, НМД *1024* Гб. Монитор *1920x1080*

5. **Программное обеспечение:**

   Операционная система семейства: *linux*, наименование: *ubuntu*, версия *20.04.3 LTS*
   интерпретатор команд: *bash* версия *4.4.19*.
   Система программирования -- версия --, редактор текстов *emacs* версия *25.2.2*
   Утилиты операционной системы --
   Прикладные системы и программы --
   Местонахождение и имена файлов программ и данных на домашнем компьютере --

6. Идея, метод, алгоритм

Для выполнения требуемых операций над деревом используются отдельно написанные функции и процедуры:

- creating_a_nod — создание корня дерева
- deliting_a_tree — удаление дерева
- print_parent — вывод родителя у вершины
- print_tree — вывод дерева
- node_search — поиск узла
- entering_a_node — добавление узла
- min_node — поиск узла с минимальным ключом
- deliting_a_node — удаление узла
- is_symmetric — проверка дерева на симметричности, используя вспомогательную функцию is_similaity

7. Сценарий выполнения работы

Program start

Creating a new tree, enter the root key:
1
The tree was created.

What you want to do?
1: Add new element
2: Print tree
3: Delete element
4: Output whether the tree is symmetric
5: Exit
1
Adding a new node, enter the key:
2
What you want to do?

1: Add new element
2: Print tree
3: Delete element
4: Output whether the tree is symmetric
5: Exit
1
Adding a new node, enter the key:
0
What you want to do?
1: Add new element
2: Print tree
3: Delete element
4: Output whether the tree is symmetric
5: Exit
1
Adding a new node, enter the key:
-1
What you want to do?
1: Add new element
2: Print tree
3: Delete element
4: Output whether the tree is symmetric
5: Exit
1
Adding a new node, enter the key:
3
What you want to do?
1: Add new element
2: Print tree
3: Delete element
4: Output whether the tree is symmetric
5: Exit
2

1
|=2
| |=3
|=0
| |=-1

What you want to do?
1: Add new element
2: Print tree
3: Delete element
4: Output whether the tree is symmetric
5: Exit
4
True

What you want to do?
1: Add new element
2: Print tree
3: Delete element
4: Output whether the tree is symmetric
5: Exit
3
Enter the key:
2
What you want to do?
1: Add new element
2: Print tree
3: Delete element
4: Output whether the tree is symmetric
5: Exit
2

1
|=3
|=0
| |=-1

What you want to do?

1: Add new element
2: Print tree
3: Delete element
4: Output whether the tree is symmetric
5: Exit
4
False

What you want to do?
1: Add new element
2: Print tree
3: Delete element
4: Output whether the tree is symmetric
5: Exit
5
End of program.

8. Распечатка протокола

main.c

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

bool is_int(const char *str) {
    while (*str) {
        if ((*str < '0' || *str > '9') && *str != '-' && *str != '.')
            return false;
        str++;
    }
    return true;
}

int main() {
    node *n;
    bool program_works = true;
    bool tree_not_created = true;
    printf("Program start\n");
    while (program_works) {
        if (tree_not_created) {
            printf("\n");
            int root;
            char c[] = "";
            bool value_is_incorrect = true;
            while (value_is_incorrect) {
                printf("Creating a new tree, enter the root key:\n");
                scanf("%s", c);
                if (is_int(c)) {
                    root = atoi(c);
                    value_is_incorrect = false;
                } else {
                    printf("Error, is not number!\n");
                }
            }
            n = creating_a_node(root);
            printf("The tree was created.\n\n");
            tree_not_created = false;
        } else {
            char menu[] = "0";
            while (!strcmp("0", menu)) {
                printf("What you want to do?\n1: Add new element\n2: Print tree\n3: Delete element\n4: Output whether the tree
is symmetric\n5: Exit\n");
                scanf("%s", menu);
                if (!strcmp("1", menu)) {
                    int key;
                    char c[] = "";
                    bool value_is_incorrect = true;
                    while (value_is_incorrect) {
                        printf("Adding a new node, enter the key:\n");
                        scanf("%s", c);
                        if (is_int(c)) {
                            key = atoi(c);
                            value_is_incorrect = false;
                        } else {
                            printf("Error, is not number!\n");
                        }
                    }
                    entering_a_node(n, key);
                } else if (!strcmp("2", menu)) {
                    printf("\n");
                    print_tree(n, 0);
```

```
                printf("\n");
            } else if (!strcmp("3", menu)) {
                int key;
                char c[] = "";
                bool value_is_incorrect = true;
                while (value_is_incorrect) {
                    printf("Enter the key:\n");
                    scanf("%s", c);
                    if (is_int(c)) {
                        key = atoi(c);
                        value_is_incorrect = false;
                    } else {
                        printf("Error, is not number!\n");
                    }
                }
                if (!node_search(n, key)) {
                    printf("Error, tree haven't this element!\n");
                } else {
                    if (n->key == key) {
                        if (!n->right && !n->left) {
                            deleting_a_tree(n);
                            n = NULL;
                            tree_not_created = true;
                        } else {
                            n = deleting_a_node(n, key);
                        }
                    } else {
                        n = deleting_a_node(n, key);
                    }
                }
            } else if (!strcmp("4", menu)) {
                if (is_symmetric(n)) {
                    printf("True\n\n");
                } else {
                    printf("False\n\n");
                }
            } else if (!strcmp("5", menu)) {
                printf("End of program.\n");
                program_works = false;
            } else {
                printf("Error, incorrect input!\n");
            }
        }
    }
    if (!tree_not_created) {
        deleting_a_tree(n);
    }
    return 0;
}
```

bin_tree.c

```
#include "bin_tree.h"

node *creating_a_node(int key) {
    node *n = (node *) malloc(sizeof(node));
    n->key = key;
    n->left = NULL;
    n->right = NULL;
    return n;
}

void deleting_a_tree(node *n) {
    printf("The tree has been removed\n");
    if (n->left)
        deleting_a_tree(n->left);
```

```
    if (n->right)
        deleting_a_tree(n->right);
    free(n);
}

void print_parent(node *n, int deep) {
    if (deep != 0) {
        printf("=%d", n->key);
    } else {
        printf("%d", n->key);
    }
}

void print_tree(node *n, int deep) {
    for (int i = 0; i < deep - 1; ++i) {
        printf("| ");
    }
    if (deep != 0)
        printf("|");
    print_parent(n, deep);
    printf("\n");
    int deep_copy = deep;
    if (n->right) {
        print_tree(n->right, ++deep_copy);
    }
    deep_copy = deep;
    if (n->left) {
        print_tree(n->left, ++deep_copy);
    }
}

node *node_search(node *n, int key) {
    if (n == NULL || n->key == key) {
        return n;
    }
    if (key < n->key) {
        return node_search(n->left, key);
    } else {
        return node_search(n->right, key);
    }
}

void entering_a_node(node *n, int key) {
    if (node_search(n, key)) {
        printf("Error, element already created\n");
    } else {
        if (n->key > key) {
            if (!n->left)
                n->left = creating_a_node(key);
            else
                entering_a_node(n->left, key);
        } else {
            if (!n->right)
                n->right = creating_a_node(key);
            else
                entering_a_node(n->right, key);
        }
    }
}

node *min_node(node *n) {
    if (!n->left) {
        return n;
    }
    return min_node(n->left);
}
```

```
node *deleting_a_node(node *n, int key) {
   if (n->key > key) {
      n->left = deleting_a_node(n->left, key);
   } else if (n->key < key) {
      n->right = deleting_a_node(n->right, key);
   } else {
      if (n->right && n->left) {
         n->key = min_node(n->right)->key;
         n->right = deleting_a_node(n->right, n->key);
      } else if (n->right) {
         node *t = n->right;
         free(n);
         n = t;
      } else if (n->left) {
         node *t = n->left;
         free(n);
         n = t;
      } else {
         free(n);
         n = NULL;
      }
   }
   return n;
}

bool is_symmetric(node *n) {
   if (n->left && n->right) {
      return is_similarity(n->left, n->right);
   } else if (!n->left && !n->right){
      return true;
   } else {
      return false;
   }
}

bool is_similarity(node *root1, node *root2) {
   if (((!root1->left && !root2->right) || (root1->left && root2->right)) &&
      ((!root1->right && !root2->left) || (root1->right && root2->left))) {
      if (root1->left && root1->right && root2->left && root2->right) {
         return is_similarity(root1->left, root2->right) && is_similarity(root1->right, root2->left);
      } else if (root1->left) {
         return is_similarity(root1->left, root2->right);
      } else if (root1->right) {
         return is_similarity(root1->right, root2->left);
      } else {
         return true;
      }
   } else {
      return false;
   }
}
```

bin_tree.h

```
#ifndef bin_tree_h
#define bin_tree_h
#include <stdio.h>
#include "stdlib.h"
#include "stdbool.h"

typedef struct node {
   struct node* left;
   struct node* right;
   int key;
} node;

node * creating_a_node(int key);
```

```
void deleting_a_tree(node * n);
void print_parent(node *n, int deep);
void print_tree(node *n, int deep);
node * node_search(node *n, int key);
void entering_a_node(node *n, int key);
node * min_node(node *n);
node * deleting_a_node(node *n, int key);
bool is_symmetric(node *n);
bool is_similarity(node *n1, node *n2);
#endif
```

9. Дневник отладки

№	Лаб или дом	Дата	Время	Событие	Действие по исправлению	Примечание

10. Замечания автора

11. 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 _____