**Отчет по лабораторной работе №23** по курсу Алгоритмы и структуры данных

Студент группы М8О-107Б-2022 Калуцкий Максим Витальевич, № по списку 13

Контакты e-mail max.kalutskij17@gmail.com

Работа выполнена: «09» .05.2023г.

Преподаватель: Аносова Наталья Павловна каф. 806

Входной контроль знаний с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Отчет сдан « » \_\_\_\_\_\_\_\_\_202 \_\_ г., итоговая оценка \_\_\_\_\_

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

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

1. **Цель работы:**  Научиться работать с динамическими структурами данных и обрабатывать деревья
2. **Задание (вариант № 13):** проверить, является ли дерево линейным списком
3. **Оборудование** (лабораторное):

*Оборудование ПЭВМ студента, если использовалось:*

Процессор Intel Core i3-4005U CPU с ОП 8096 Мб, НМД \_ 131072\_ Мб. Монитор lenovo

Другие устройства \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

*Программное обеспечение ЭВМ студента, если использовалось:*

Операционная система семейства \_\_\_Unix\_\_\_\_, наименование \_\_ Ubuntu \_\_\_\_\_\_\_\_\_\_\_\_ версия 4.4.2

интерпретатор команд \_\_\_\_bash\_\_\_\_\_\_\_\_ версия \_\_5.0.17\_\_\_\_.

Система программирования \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_версия \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Редактор текстов \_\_\_\_\_nano\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ версия \_25.2.2 \_\_\_\_\_\_\_\_\_

Утилиты операционной системы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Прикладные системы и программы **\_\_\_**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Местонахождение и имена файлов программ и данных на домашнем компьютере \_\_\_\_\_home/jekich228\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **Идея, метод, алгоритм** решения задачи(в формах:словесной,псевдокода,графической[блок-схема,диаграмма,рисунок,таблица] или формальные спецификации с пред- и постусловиями)

Я решил реализовать бинарное упорядоченное дерево в рамках этой лабараторной работы. Для реализации узла дерева я использовать следующую структуру:

struct elem\_tree{

    int value;

    elem\_tree \* right;

    elem\_tree \* left;

};

В каждом узле у меня будет храниться 3поля: поле со значением узла, поле с указателем на правого сына и поле с указателем на левого сына.

Мне в любой момент времени будет доступен только указатель на корень дерева, через который я смогу передвигаться между узлами.

Добавление нового узла реализовано у меня следующим образом: я динамически выделяю память для узла, после чего передаю указатель только что созданного узла в функцию, которая помещает узел по всем правилам бинарного упорядоченного дерева, либо делает узел корнем дерева, если дерево пустое.

Визуализация же дерева была сделана рекурсивно, начиная с самого правого нижнего ребенка, двигаясь к левому нижнему. То есть структура расположения узлов дерева обеспечивается за счёт табуляции выведенных строк.

Для удаления узла была такая логика: для начала мы опускаемся по дереву туда, где находится узел для удаления, причем я смотрю через один узел, чтобы после удаления узла было легко перевязать дерево, прикрепив оставшиеся узлы по правилам построения бинарного упорядоченного дерева.

В основной части программы будем использовать меню, в котором есть 6 опций:

1. Добавление нового узла в дерево, вне зависимости пустое оно или нет (Add). Запрашивает значение корня дерева, а затем создает узел, вызывая функцию add и передавая туда указатель, который возвращает функция new\_top(), также в add нужно передать указатель на корень дерева: add(new\_top(s),head);
2. Удаление узла дерева (Del).Вызывает функцию delete\_top(head,s), передавая ей указатель на корень дерева, а также значение узла, который нужно удалить.
3. Выполнение задания (проверка на линейный список) (Check). Вызывает функцию task.
4. Печать дерева (View). Вызывает функцию printTree (head), передавая указатель на корень дерева.
5. Выход (Exit). Выходит из меню.
6. **Сценарий выполнения работы** [план работы,первоначальный текст программы в черновике(можно на отдельном листе)итесты либо соображения по тестированию].

Сама программа:

#include <bits/stdc++.h>

using namespace std;

struct elem\_tree{

    int value;

    elem\_tree \* right;

    elem\_tree \* left;

};

elem\_tree\* new\_top(int value){

    elem\_tree \* kek = new elem\_tree;

    kek->value = value;

    kek->right = NULL;

    kek->left = NULL;

    return kek;

}

void add(elem\_tree\* elem,  elem\_tree \* & head){

    if(head == NULL) head = elem;

    else{

        if(head->value > elem->value){

            if(head->left==NULL)head->left = elem;

            else add(elem,head->left);

        }

        else if(head->value == elem->value) { cout<<"такой узел уже ѝущеѝтвует"<<endl;}

        else{

            if(head->right==NULL) head->right = elem;

            else add(elem,head->right);

        }

    }

}

void printTree(elem\_tree\* root, int indent = 0) {

    if (root != NULL) {

        // Выводим правое поддерево

        printTree(root->right, indent + 4);

        // Выводим текущий узел

        cout << string(indent, ' ') << root->value << endl;

        // Выводим левое поддерево

        printTree(root->left, indent + 4);

    }

}

void delete\_top(elem\_tree \* & phead, int value){

    elem\_tree \* head = phead;

    elem\_tree \* lol = phead->right;

    elem\_tree \* headr = phead->right;

    elem\_tree \* headl = phead->left;

        if(head->value < value){

            if(((head->right)->value) == value){

                if((head->right)->left == NULL and (head->right)->right == NULL){

                delete((head->right));

                head->right = NULL;

            }

            else if(((head->right)->left == NULL and (head->right)->right != NULL) or ((head->right)->left != NULL and (head->right)->right == NULL)){

                if(((head->right)->left == NULL and (head->right)->right != NULL)) head->right = (head->right)->right;

                else head->right = (head->right)->left;

                delete(headr);

            }

            else{

                elem\_tree \* kek = head->right;

                head->right = (head->right)->right;

                (head->right)->left = kek->left;

                delete(kek);

            }

            }

            else delete\_top(head->right,value);

        }

        else if(head->value > value){

            if(((head->left)->value) == value){

                if((head->left)->left == NULL and (head->left)->right == NULL){

                    delete((head->left));

                    head->left = NULL;

            }

            else if(((head->left)->left == NULL and (head->left)->right != NULL) or ((head->left)->left != NULL and (head->left)->right == NULL)){

                if(((head->left)->left == NULL and (head->left)->right != NULL)) head->left = (head->left)->right;

                else head->left = (head->left)->left;

                delete(headl);

            }

            else{

                elem\_tree \* kek = head->left;

                head->left = (head->left)->right;

                (head->left)->left = kek->left;

                delete(kek);

                }

            }

            else delete\_top(head->left,value);

        }

        else if(phead -> value == value){

            if(phead->right ==NULL){

                elem\_tree \* keke = phead->left;

                delete(phead);

                phead = keke;

                delete(keke);

            }

            else if(phead->left ==NULL){

                elem\_tree \* keke = phead->right;

                delete(phead);

                phead = keke;

                delete(keke);

            }

            else{

               while((lol->left) != NULL) lol = (lol->left);

               lol->left = phead->left;

               phead = phead->right;

            }

        }

        else cout<<"Введены некорректные данные"<<endl;

}

bool is\_linear\_list\_of\_vertices(elem\_tree \* phead){

    if(phead->left != NULL and phead->right == NULL){

        is\_linear\_list\_of\_vertices( phead->left);

    }

    else if(phead->left == NULL and phead->right != NULL){

        is\_linear\_list\_of\_vertices( phead->right);

    }

    else if(phead->left == NULL and phead->right == NULL)return 1;

    else return 0;

}

int main()

{

    elem\_tree \* head = NULL;

    while(1){

        cout<<"Ведите команду:"<<endl;

        cout<<"Add - добавление нового узла."<<endl;

        cout<<"View - визуализациѝ дерева."<<endl;

        cout<<"Del - удаление узла."<<endl;

        cout<<"Break - завершить работу программы."<<endl;

        cout<<"Check - проверка, ѝвлѝетѝѝ ли дерево линейным ѝпиѝком вершин."<<endl;

        string k;

        cin>>k;

        if(k == "Add"){

            cout<<"Введите значние узла."<<endl;

            int k;

            int s;

            cin>>s;

            if(head == NULL) cout<<"Корень дерева ѝоздан!"<<endl;

            add(new\_top(s),head);

        }

        if(k == "View"){

            printTree(head);

        }

        if(k == "Break"){

            break;

        }

        if(k == "Del"){

            cout<<"Введите значение удалѝемого узла.";

            int s;

            cin>>s;

            delete\_top(head,s);

        }

        if(k == "Check"){

            if(is\_linear\_list\_of\_vertices(head)) cout<<"Дерево ЯВЛЯЕТСЯ линейным ѝпиѝком вершин."<<endl;

            else cout<<"Дерево НЕ ЯВЛЯЕТСЯ линейным ѝпиѝком вершин."<<endl;

        }

    }

}

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

PS C:\Users\heleu\Desktop\laba\_2\_sem\_kek> & 'c:\Users\heleu\.vscode\extensions\ms-vscode.cpptools-1.15.4-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-ecyd2cal.qsk' '--stdout=Microsoft-MIEngine-Out-rnwyvt41.x0m' '--stderr=Microsoft-MIEngine-Error-50hbueki.4gx' '--pid=Microsoft-MIEngine-Pid-0mln25ll.uqx' '--dbgExe=C:\MinGW\bin\gdb.exe' '--interpreter=mi'

Ведите команду:

Add - добавление нового узла.

View - визуализациѝ дерева.

Del - удаление узла.

Break - завершить работу программы.

Check - проверка, ѝвлѝетѝѝ ли дерево линейным ѝпиѝком вершин.

Add

Введите значние узла.

150

Корень дерева ѝоздан!

Ведите команду:

Add - добавление нового узла.

View - визуализациѝ дерева.

Del - удаление узла.

Break - завершить работу программы.

Check - проверка, ѝвлѝетѝѝ ли дерево линейным ѝпиѝком вершин.

Add

Введите значние узла.

200

Ведите команду:

Add - добавление нового узла.

View - визуализациѝ дерева.

Del - удаление узла.

Break - завершить работу программы.

Check - проверка, ѝвлѝетѝѝ ли дерево линейным ѝпиѝком вершин.

Add

Введите значние узла.

100

Ведите команду:

Add - добавление нового узла.

View - визуализациѝ дерева.

Del - удаление узла.

Break - завершить работу программы.

Check - проверка, ѝвлѝетѝѝ ли дерево линейным ѝпиѝком вершин.

Add

Введите значние узла.

220

Ведите команду:

Add - добавление нового узла.

View - визуализациѝ дерева.

Del - удаление узла.

Break - завершить работу программы.

Check - проверка, ѝвлѝетѝѝ ли дерево линейным ѝпиѝком вершин.

Add

Введите значние узла.

90

Ведите команду:

Add - добавление нового узла.

View - визуализациѝ дерева.

Del - удаление узла.

Break - завершить работу программы.

Check - проверка, ѝвлѝетѝѝ ли дерево линейным ѝпиѝком вершин.

Add

Введите значние узла.

75

Ведите команду:

Add - добавление нового узла.

View - визуализациѝ дерева.

Del - удаление узла.

Break - завершить работу программы.

Check - проверка, ѝвлѝетѝѝ ли дерево линейным ѝпиѝком вершин.

View

220

200

150

100

90

75

Ведите команду:

Add - добавление нового узла.

View - визуализациѝ дерева.

Del - удаление узла.

Break - завершить работу программы.

Check - проверка, ѝвлѝетѝѝ ли дерево линейным ѝпиѝком вершин.

Del

Введите значение удалѝемого узла.150

Ведите команду:

Add - добавление нового узла.

View - визуализациѝ дерева.

Del - удаление узла.

Break - завершить работу программы.

Check - проверка, ѝвлѝетѝѝ ли дерево линейным ѝпиѝком вершин.

View

220

200

100

90

75

Ведите команду:

Add - добавление нового узла.

View - визуализациѝ дерева.

Del - удаление узла.

Break - завершить работу программы.

Check - проверка, ѝвлѝетѝѝ ли дерево линейным ѝпиѝком вершин.

Break

PS C:\Users\heleu\Desktop\laba\_2\_sem\_kek>

1. **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 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Выводы**: Проделав данную работу, я научился составлять программы на языке Си++ для построения и обработки деревьев общего вида или упорядоченного двоичного дерева, содержащего узлы различных типов.

Недочёты при выполнении задания могут быть устранены следующим образом: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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