Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра автоматизированных систем управления (АСУ)

ДЕРЕВЬЯ

Отчет по лабораторной работе №4

По дисциплине

«Структуры и алгоритмы обработки данных в ЭВМ»

Студент гр. 431-3

_____ А.В. Гурулёв

«___»_____ 2022 г.

Проверил: профессор кафедры АСУ, д.т.н.

_____ А.Н. Горитов

«___»_____ 2022 г.

Томск 2022

1 Задание на лабораторную работу

Напишите программу, которая формирует бинарное дерево поиска, выводит построенное дерево на экран и подсчитывает сумму элементов из всех листьев дерева. Данные могут вводиться с клавиатуры, из файла или генерироваться с помощью генератора случайных чисел. Выбор способа данных выполняется во время работы программы.

Для реализации АТД “Дерево” используйте динамическое распределение памяти. Перед завершением работы программы освободить занимаемую динамическую память. Для этого используйте поэлементное удаление элементов динамической структуры данных.

2 Алгоритм решения задачи

1. Инициализируем дерево;
2. Запрашиваем у пользователя режим работы;
3. Если пользователь выбрал случайные значения, то заполняем дерево случайными значениями; Если выбрал чтение с файла: Открываем файл; Заносим данные в дерево; Если выбрал ввод с клавиатуры, то запрашиваем количество чисел; Заносим числа в дерево;
4. Считаем сумму листков при помощи симметричного обхода;
5. Выводим дерево;
6. Очищаем память;

3 Листинг программы

Для main.cpp:

```
#include <iostream>
#include <string>
#include <fstream>
#include <Windows.h>
#include <ctime>
#include "MyTree.h"

using namespace std;
using namespace MyTask;
```

```
MyTree* RandomData()
{
  MyTree* tree = new MyTree;
  for (int i = 0; i < 20; i++)
  {
        (*tree).insert_tree(&(*tree).root, (rand() % 100), (*tree).root);
  }
  return tree;
}

MyTree* InputData()
{
  MyTree* tree = new MyTree;
  int in = 0;
  int size = 0;

  cout << "\n\tВвыедите кол-во элементов: ";
  cin >> size;
  cout << "\n\tВвыедите элементы: \n";

  for (int i = 0; i < size; i++)
  {
        cin >> in;
        (*tree).insert_tree(&(*tree).root, in, (*tree).root);
  }
  return tree;
}

MyTree* fileData()
```

```
{
 MyTree* tree = new MyTree;
 int integer = 0;

 ifstream in;
 in.open("f.txt");

 while (in >> integer)
 {
        (*tree).insert_tree(&(*tree).root, integer, (*tree).root);
 }
 return tree;
}



int main()
{
 setlocale(0, "");
 srand(time(NULL));

 cout << "Выберите режим:\n\t1: Рандом\n\t2: Ввод в консоли\n\t3: Чтение с файла\nВаш выбор: ";

 __int16 mode = 0;
 cin >> mode;

 MyTree* tree;

 switch (mode)
```

```
{
case 1:
        tree = RandomData();
        break;
case 2:
        tree = InputData();
        break;
case 3:
        tree = fileData();
        break;
default:
        cout << "Некорректный ввод";
        exit(EXIT_FAILURE);
        break;
}

cout << "\n\nДерево:\n";

tree->travel_tree_print_boring(tree->root);

cout << endl;

cout << "Сумма листьев: " << tree->travel_tree_sum(tree->root) << endl;

cout << endl;

system("pause");

return 0;
}
```

Для MyTree.cpp:

```
#include <iostream>
#include <Windows.h>
#include "MyTree.h"

using namespace std;
using namespace MyTask;

//Инициализация
MyTree::MyTree()
{
  root = NULL;
}

//Очитка памяти
MyTree::~MyTree()
{
  travel_tree_clear(root);
}

//Поиск элемента
MyTree::Tree* MyTree::search_tree(Tree* node, int info)
{
  //Если узел пуст - прекращаем обход
  if (!node)
        return NULL;

  //Если нашли нужные данные - возвращаем узел
  if (node->item == info)
        return node;

  //Если нужный элемент меньше текущего - идём в левую ветвь иначе - в правую
  if (info < node->item)
        return(search_tree(node->left, info));
```

```
  else
        return(search_tree(node->right, info));
}

//Обход дерева(Вычсиление суммы листьев)
int MyTree::travel_tree_sum(Tree* node)
{
  if (node)
  {
        int sum = 0;
        if ((!node->left) && (!node->right))
              sum += node->item;
        sum += travel_tree_sum(node->left);
        sum += travel_tree_sum(node->right);
        return sum;
  }
  return 0;
}

//Обход дерева(Вывод в консоль)
void MyTree::travel_tree_print_boring(Tree* node, int level)
{
  if (node)
  {
        if ((node->left))
              travel_tree_print_boring(node->left, level + 1);

        if (node != root)
        {
              for (int i = 0; i < level; i++)
              {
                    cout << "\t";
              }
        }
        cout << node->item;
```

```
            cout << "\n";
            if (node->right)
            {
                  travel_tree_print_boring(node->right, level + 1);
            }
  }
}
//Обход дерева(Очистка памяти)
void MyTree::travel_tree_clear(Tree* node)
{
  if (node)
  {
            travel_tree_clear(node->left);
            travel_tree_clear(node->right);
            node->left = NULL;
            node->right = NULL;
            node->parent = NULL;
            node->item = 0;
            delete[] node;
  }
}

//Получение максимального элемента
MyTree::Tree* MyTree::max_of_tree(Tree* node)
{
  Tree* max;
  if (!node)
            return NULL;
  max = node;
  while (max->right)
            max = max->right;
  return max;
}

//Получение минимального элемента
```

```
MyTree::Tree* MyTree::min_of_tree(Tree* node)
{
  Tree* min;
  if (!node)
        return NULL;
  min = node;
  while (min->left)
        min = min->left;
  return min;
}

//Вставка
void MyTree::insert_tree(Tree** node, int info, Tree* parent)
{
  Tree* timeNode;
  if (!(*node))
  {
        timeNode = new Tree;
        timeNode->item = info;
        timeNode->left = timeNode->right = NULL;
        timeNode->parent = parent;
        *node = timeNode;
        if (!parent)
              root = *node;
        return;
  }

  if (info < (*node)->item)
        insert_tree(&((*node)->left), info, *node);
  else
        if (info != (*node)->item)
              insert_tree(&((*node)->right), info, *node);
}

//Удаление
```

```
void MyTree::remove_tree(Tree* node)
{
  if (node->left == node->right == NULL)
  {
        if (node->parent->left == node)
        {
              node->parent->left = NULL;
        }
        else
        {
              node->parent->right = NULL;
        }

        node->item = 0;
        node->left = NULL;
        node->right = NULL;
        delete[] node;

        return;
  }

  Tree* pOne = node->left;
  Tree* pTwo = node;

  while (pOne)
  {
        pOne = pOne->right;
        pTwo = pOne;
  }

  if (node->parent->left == node)
  {
        node->parent->left = pTwo;
  }
  else
```

```
  {
        node->parent->right = pTwo;
  }

  node->item = 0;
  node->left = NULL;
  node->right = NULL;
  delete[] node;

  return;
}
```

Для MyTree.h:

```
namespace MyTask
{
  class MyTree
  {
  public:
        //Структура
        struct Tree
        {
              int item;
              Tree* parent;
              Tree* left;
              Tree* right;
        };

        //Корень
        Tree* root;

        //Инициализация
        MyTree();

        //Очитка памяти
        ~MyTree();
```

```
        //Поиск элемента
        Tree* search_tree(Tree* node, int info);

        //Обход дерева(Вычсиление суммы листьев)
        int travel_tree_sum(Tree* node);

        //Обход дерева(Вывод в консоль)
        void travel_tree_print_boring(Tree* node, int level = 0);

        //Обход дерева(Очистка памяти)
        void travel_tree_clear(Tree* node);

        //Получение максимального элемента
        Tree* max_of_tree(Tree* node);

        //Получение минимального элемента
        Tree* min_of_tree(Tree* node);

        //Вставка
        void insert_tree(Tree** node, int info, Tree* parent);

        //Удаление
        void remove_tree(Tree* node);
    };
}
```

4 Пример решения

На рисунке 4.1 можно увидеть пример результата программы в 3 режиме. Вывод состоит из выведенного дерева и результата суммы его листьев.

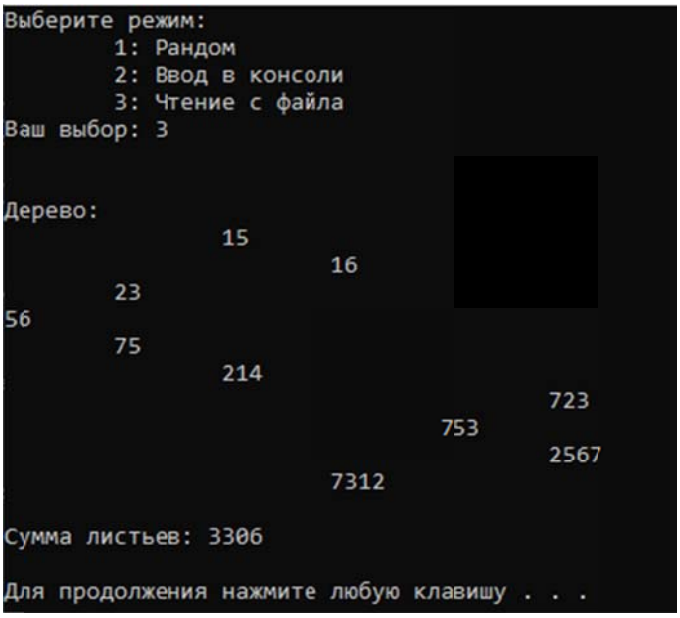

Рисунок 4.1 - Результат выполнения программы

На рисунке 4.2 можно увидеть входные данные файла.

Рисунок 4.2 - Входные данные

5 Вывод

Я изучил как устроен АТД “Дерево”, его основные методы, так же как с ними работать.