Министерство науки и высшего образования Российской Федерации
Калужский филиал
федерального государственного бюджетного
образовательного учреждения высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(КФ МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ **ИУК «Информатика и управление»**

КАФЕДРА **ИУК4 «Программное обеспечение ЭВМ, информационные технологии»**

ЛАБОРАТОРНАЯ РАБОТА №2

«Создание и обработка древовидных структур данных»

ДИСЦИПЛИНА: «Типы и структуры данных»

Выполнил: студент гр. ИУК4-31Б _____ (Суриков Н. С.)
(подпись) (Ф.И.О.)

Проверил: _____ (Былинка М. И.)
(подпись) (Ф.И.О.)

Дата сдачи (защиты):

Результаты сдачи (защиты):

- Балльная оценка:
- Оценка:

Калуга, 2024

Цель: формирование практических навыков создания алгоритмов обработки древовидных структур данных.

Задачи:

1. Изучить виды деревьев.
2. Научиться стоить двоичные деревья, деревья поиска.
3. Изучить способы балансировки деревьев.
4. Познакомиться с основными алгоритмами обработки деревьев.
5. Реализовать основные алгоритмы обработки древовидных структур данных (создание, удаление, поиск, добавление и удаление элемента), а также алгоритм согласно полученному варианту.

Вариант 25

Формулировка задания:

1. Разработать консольное приложение, написанное с помощью объектно-ориентированной технологии. Индивидуальное задание предусмотрено вариантом, который назначает преподаватель.
2. Приложение необходимо запускать для демонстрации из командной строки с указанием названий приложения и трех файлов:
   − все входные данные (например, последовательности чисел, коэффициенты многочленов и т.д.) считать из первого файла;
   − все выходные данные записать во второй файл;
   − все возникшие ошибки записать в третий файл – файл ошибок.
3. Все основные сущности приложения представить в виде отдельных классов.
4. Необходимо предусмотреть пользовательское меню, содержащее набор команд всех основных операций для работы со списком, а также команду для запуска индивидуального задания.
5. В приложении также должны быть учтены все критические ситуации, обработанные с помощью класса исключений.

Индивидуальное задание:

Построить двоичное дерево из целых чисел и написать следующие процедуры:

a) вывод элементов дерева по уровням;

b) удаление из дерева отрицательных элементов.

Листинг программы:

`main.cpp`

```
  1  #include "CMenu.h"
  2  #include "MyError.h"
  3  #include <iostream>
  4
  5  CMenu::CMenu() {}
  6   CMenu::CMenu(std::string title, CMenuItem *items, size_t count) : m_title(title), m_items(items),
m_count(count) {}
  7
  8  int CMenu::getSelect() const
  9  {
 10      return m_select;
 11  }
 12
 13  bool CMenu::getRunning()
 14  {
 15      return m_running;
 16  }
 17  void CMenu::setRunning(bool _running)
 18  {
 19      m_running = _running;
 20  }
 21
 22  size_t CMenu::getCount() const
 23  {
 24      return m_count;
 25  }
 26
 27  std::string CMenu::getTitle()
 28  {
 29      return m_title;
 30  }
 31
 32  CMenuItem *CMenu::getItems()
 33  {
 34      return m_items;
 35  }
 36
 37  void CMenu::print()
 38  {
 39
 40      for (size_t i{}; i < m_count - 1; ++i)
 41      {
 42          std::cout << i + 1 << ". ";
 43          m_items[i].print();
 44          std::cout << std::endl;
 45      }
 46      std::cout << "0. ";
 47      m_items[m_count - 1].print();
 48      std::cout << std::endl;
 49  }
 50
```

```cpp
void CMenu::printTitle()
{
    std::cout << "\033[2J\033[1;1H";
    std::cout << "\t" << m_title << std::endl;
}

int CMenu::runCommand()
{
    printTitle();
    print();
    std::cout << "\n   Select >> ";
    std::string SelectInput;
    bool flag = true;
    std::cin >> SelectInput;
    for (int i{0}; i < SelectInput.size(); i++)
    {
        if (!(SelectInput[i] >= '0' && SelectInput[i] <= '9'))
        {
            flag = false;
        }
    }
    if (flag)
    {
        m_select = std::stoi(SelectInput);
    }
    else
    {
        std::cout << "\033[2J\033[1;1H";
        throw MyError{"Wrong input. Enter only number."};

        std::cout << "Нажмите Enter, чтобы продолжить...";
        std::cin.clear();
        std::cin.ignore(1024, '\n');
        std::cin.get();
        std::cin.clear();
        std::cin.ignore(1024, '\n');

        return 1;
    }
    if (m_select == 0)
    {
        return m_items[m_count - 1].run();
    }
    else
    {
        if ((m_select > m_count - 1) || (m_select < 0))
        {
            std::cout << "\033[2J\033[1;1H";
            throw MyError{"Wrong input. Enter correct number of menu."};

            std::cout << "Нажмите Enter, чтобы продолжить...";
            std::cin.clear();
            std::cin.ignore(1024, '\n');
            std::cin.get();
            std::cin.clear();
            std::cin.ignore(1024, '\n');

            return 1;
        }
        else
        {
            std::cout << "\033[2J\033[1;1H";
            return m_items[m_select - 1].run();
        }
    }
}
```

Cmenu.h

```cpp
#ifndef MYMENU_CMENU_H
#define MYMENU_CMENU_H

#include "CMenuItem.h"
#include <cstddef>

class CMenu
{
public:
    CMenu();
    CMenu(std::string, CMenuItem *, size_t);
    int getSelect() const;
    bool getRunning();
    void setRunning(bool);
    std::string getTitle();
    size_t getCount() const;
    CMenuItem *getItems();
    void print();
    void printTitle();
    int runCommand();

private:
    int m_select{-1};
    size_t m_count{};
    bool m_running{true};
    std::string m_title{};
    CMenuItem *m_items{};
};

#endif // MYMENU_CMENU_H
```

Cmenu.cpp

```cpp
#include "CMenu.h"
#include "MyError.h"
#include "Tools.h"
#include <iostream>

namespace ExpressionTree
{
    CMenu::CMenu() {}
    CMenu::CMenu(std::string title, CMenuItem *items, size_t count) : m_title(title), m_items(items), m_count(count) {}

    int CMenu::getSelect() const
    {
        return m_select;
    }

    bool CMenu::getRunning()
    {
        return m_running;
    }
    void CMenu::setRunning(bool _running)
    {
        m_running = _running;
    }

    size_t CMenu::getCount() const
    {
        return m_count;
    }

    std::string CMenu::getTitle()
    {
        return m_title;
```

```
33      }
34
35      CMenuItem *CMenu::getItems()
36      {
37          return m_items;
38      }
39
40      void CMenu::print()
41      {
42
43          for (size_t i{}; i < m_count - 1; ++i)
44          {
45              std::cout << i + 1 << ". ";
46              m_items[i].print();
47              std::cout << std::endl;
48          }
49          std::cout << "0. ";
50          m_items[m_count - 1].print();
51          std::cout << std::endl;
52      }
53
54      void CMenu::printTitle()
55      {
56          ClearScreen();
57          std::cout << "\t" << m_title << std::endl;
58      }
59
60      int CMenu::runCommand()
61      {
62          printTitle();
63          print();
64          std::cout << "\n   Select >> ";
65          std::string SelectInput;
66          bool flag = true;
67          std::cin >> SelectInput;
68          for (int i{0}; i < SelectInput.size(); i++)
69          {
70              if (!(SelectInput[i] >= '0' && SelectInput[i] <= '9'))
71              {
72                  flag = false;
73              }
74          }
75          if (flag)
76          {
77              m_select = std::stoi(SelectInput);
78          }
79          else
80          {
81              ClearScreen();
82              throw MyError{"Wrong input. Enter only number."};
83              WaitForEnter();
84              return 1;
85          }
86          if (m_select == 0)
87          {
88              return m_items[m_count - 1].run();
89          }
90          else
91          {
92              if ((m_select > m_count - 1) || (m_select < 0))
93              {
94                  ClearScreen();
95                  throw MyError{"Wrong input. Enter correct number of menu."};
96                  WaitForEnter();
97                  return 1;
98              }
99              else
100             {
101                 ClearScreen();
102                 return m_items[m_select - 1].run();
```

```
            }
        }
    }
}
```

CmenuItem.h

```
#ifndef MYMENU_CPP_CMENUITEM_H
#define MYMENU_CPP_CMENUITEM_H
#include <string>
#include <functional>

using Func = std::function<int()>;

class CMenuItem
{
public:
    CMenuItem(std::string, Func);
    Func m_func{};
    std::string m_item_name{};
    std::string getName();
    void print();
    int run();
};

#endif // MYMENU_CPP_CMENUITEM_H
```

CmenuItem.cpp

```
#include "CMenuItem.h"
#include <iostream>

CMenuItem::CMenuItem(std::string item_name, Func func) : m_item_name(item_name), m_func(func) {}

std::string CMenuItem::getName()
{
    return m_item_name;
}

void CMenuItem::print()
{
    std::cout << m_item_name;
}

int CMenuItem::run()
{
    return m_func();
}
```

BinTree.h

```
#ifndef BINTREE_H
#define BINTREE_H

#include <fstream>
#include <iostream>
#include <queue>

namespace ExpressionTree
{

   struct Node
   {
```

```
13              int data;
14              Node *left;
15              Node *right;
16      };
17
18      class BinTree
19      {
20      public:
21              BinTree();
22              bool IsEmpty() const;
23
24              void Insert(int info);
25              void LevelOrderTraversal();
26              void RemoveNegative();
27              void PrintTree();
28              void WriteToFile(const std::string &filename);
29              void Clear();
30
31      private:
32              Node *root;
33
34              Node *Create_Node(int info);
35              Node *InsertRec(Node *node, int info);
36              Node *RemoveNegativeRec(Node *node);
37              void PrintTreeRec(Node *node, int depth);
38              void WriteToFileRec(Node *node, std::ofstream &out);
39              void ClearRec(Node *node);
40      };
41  }
42
43  #endif // BINTREE_H
```

BinTree.cpp

```
1   #include "BinTree.h"
2
3   namespace ExpressionTree
4   {
5
6      BinTree::BinTree() : root(nullptr) {}
7
8      Node *BinTree::Create_Node(int info)
9      {
10             Node *temp = new Node();
11             temp->left = nullptr;
12             temp->right = nullptr;
13             temp->data = info;
14             return temp;
15     }
16
17     bool BinTree::IsEmpty() const
18     {
19             return root == nullptr;
20     }
21
22     void BinTree::Insert(int info)
23     {
24             root = InsertRec(root, info);
25     }
26
27     void BinTree::LevelOrderTraversal()
28     {
29             if (root == nullptr)
30             {
31                     std::cout << "Дерево пустое\n";
32                     return;
33             }
```

```
34
35          std::queue<Node *> q;
36          q.push(root);
37
38          while (!q.empty())
39          {
40                  int levelSize = q.size(); // Количество узлов на текущем уровне
41                  for (int i = 0; i < levelSize; ++i)
42                  {
43                          Node *node = q.front();
44                          q.pop();
45                          std::cout << node->data << " "; // Вывод данных узла
46
47                          if (node->left)
48                                  q.push(node->left);
49                          if (node->right)
50                                  q.push(node->right);
51                  }
52                  std::cout << std::endl;
53          }
54  }
55
56  void BinTree::RemoveNegative()
57  {
58          root = RemoveNegativeRec(root);
59  }
60
61  void BinTree::PrintTree()
62  {
63          PrintTreeRec(root, 0);
64  }
65
66  void BinTree::WriteToFile(const std::string &filename)
67  {
68          std::ofstream out(filename);
69          if (out.is_open())
70          {
71                  WriteToFileRec(root, out);
72                  out.close();
73                  std::cout << "Дерево записано в файл: " << filename << std::endl;
74          }
75          else
76          {
77                  std::cerr << "Не удалось открыть файл для записи.\n";
78          }
79  }
80
81  void BinTree::Clear()
82  {
83          ClearRec(root);
84          root = nullptr;
85          std::cout << "Дерево успешно удалено.\n";
86  }
87
88  Node *BinTree::InsertRec(Node *node, int info)
89  {
90          if (node == nullptr)
91          {
92                  return Create_Node(info);
93          }
94          if (info < node->data)
95          {
96                  node->left = InsertRec(node->left, info);
97          }
98          else
99          {
100                 node->right = InsertRec(node->right, info);
101         }
102         return node;
103 }
```

```
104
105     Node *BinTree::RemoveNegativeRec(Node *node)
106     {
107             if (node == nullptr)
108                     return nullptr;
109
110             node->left = RemoveNegativeRec(node->left);
111             node->right = RemoveNegativeRec(node->right);
112
113             if (node->data < 0)
114             {
115                     Node *temp = (node->left) ? node->left : node->right;
116                     delete node;
117                     return temp;
118             }
119             return node;
120     }
121
122     void BinTree::PrintTreeRec(Node *node, int depth)
123     {
124             if (node)
125             {
126                     PrintTreeRec(node->right, depth + 1);
127                     std::cout << std::string(depth * 4, ' ') << node->data << std::endl;
128                     PrintTreeRec(node->left, depth + 1);
129             }
130     }
131
132     void BinTree::WriteToFileRec(Node *node, std::ofstream &out)
133     {
134             if (node)
135             {
136                     WriteToFileRec(node->left, out);
137                     out << node->data << std::endl;
138                     WriteToFileRec(node->right, out);
139             }
140     }
141
142     void BinTree::ClearRec(Node *node)
143     {
144             if (node)
145             {
146                     ClearRec(node->left);
147                     ClearRec(node->right);
148                     delete node;
149             }
150     }
151 }
```

MyError.cpp

```
1  #define _CRT_SECURE_NO_WARNINGS
2  #include "MyError.h"
3  #include <iostream>
4  #include <fstream>
5  #include <chrono>
6
7  namespace ExpressionTree
8  {
9      const char* MyError::getError() const
10     {
11             return m_error.c_str();
12     }
13
14
15     MyError::MyError(std::string error)
16     {
```

```
            m_error = error;
            logging();
    }

    void MyError::logging()
    {
            std::fstream file;
            auto now = std::chrono::system_clock::now();
            std::time_t end_time = std::chrono::system_clock::to_time_t(now);
            file.open(m_file, std::ios::app);
            file << "WARNING: " << m_error.c_str() << "|" << std::ctime(&end_time);
            file.close();
    }
}
```

MyError.h

```
#ifndef MYERROR_H
#define MYERROR_H
#include <string>
#include <string_view>

namespace ExpressionTree
{
    class MyError
    {
    public:
            MyError(std::string error);
            static std::string m_file;
            const char* getError() const;

    private:
            void logging();
            std::string m_error;


    };
}
#endif // MYERROR_H
```

Tools.cpp

```
#include <iostream>
#include <limits>
#include "Tools.h"



#ifdef _WIN32
#include <windows.h>
#endif

void ClearScreen() {
#ifdef _WIN32
    // Очистка экрана для Windows
    system("cls");
#else
    // Очистка экрана для Linux/Unix
    system("clear");
#endif
}

void WaitForEnter()
{
    std::cout << "Нажмите Enter для продолжения...";
```

```
24      std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
25      std::cin.get(); // Ожидание ввода
26  }
```

Результат работы:

```
        Дерево выражения
1. Распечатать дерево
2. Вывод дерева по уровням
3. Удаление дерева
4. Удаление отрицательных элементов
5. Вывод в файл
0. Выход

   Select >> 1

    499
        44
25
    3
        -5
            -36
                -323
Нажмите Enter для продолжения…

25
3 499
-5 44
-36
-323
Нажмите Enter для продолжения…

    499
        44
25
    3
Нажмите Enter для продолжения...
```

Вывод: в ходе работы были сформированы практические навыки создания алгоритмов обработки древовидных структур данных.