



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

  
(подпись)

( Суриков Н. С. )  
(Ф.И.О.)

Проверил:

  
(подпись)

( Былинка М. И. )  
(Ф.И.О.)

Дата сдачи (защиты): 5.11.24

Результаты сдачи (защиты):

- Балльная оценка:

85

- Оценка:

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

**Задачи:**

1. Изучить виды деревьев.
2. Научиться строить 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
```

```

51 void CMenu::printTitle()
52 {
53     std::cout << "\033[2J\033[1;1H";
54     std::cout << "\t" << m_title << std::endl;
55 }
56
57 int CMenu::runCommand()
58 {
59     printTitle();
60     print();
61     std::cout << "\n  Select >> ";
62     std::string SelectInput;
63     bool flag = true;
64     std::cin >> SelectInput;
65     for (int i{0}; i < SelectInput.size(); i++)
66     {
67         if (!(SelectInput[i] >= '0' && SelectInput[i] <= '9'))
68         {
69             flag = false;
70         }
71     }
72     if (flag)
73     {
74         m_select = std::stoi(SelectInput);
75     }
76     else
77     {
78         std::cout << "\033[2J\033[1;1H";
79         throw MyError{"Wrong input. Enter only number."};
80
81         std::cout << "Нажмите Enter, чтобы продолжить...";
82         std::cin.clear();
83         std::cin.ignore(1024, '\n');
84         std::cin.get();
85         std::cin.clear();
86         std::cin.ignore(1024, '\n');
87
88         return 1;
89     }
90     if (m_select == 0)
91     {
92         return m_items[m_count - 1].run();
93     }
94     else
95     {
96         if ((m_select > m_count - 1) || (m_select < 0))
97         {
98             std::cout << "\033[2J\033[1;1H";
99             throw MyError{"Wrong input. Enter correct number of menu."};
100
101             std::cout << "Нажмите Enter, чтобы продолжить...";
102             std::cin.clear();
103             std::cin.ignore(1024, '\n');
104             std::cin.get();
105             std::cin.clear();
106             std::cin.ignore(1024, '\n');
107
108             return 1;
109         }
110         else
111         {
112             std::cout << "\033[2J\033[1;1H";
113             return m_items[m_select - 1].run();
114         }
115     }
116 }

```

## Cmenu.h

```

1  #ifndef MYMENU_CMENU_H
2  #define MYMENU_CMENU_H
3
4  #include "CMenuItem.h"
5  #include <cstdint>
6
7  class CMenu
8  {
9  public:
10     CMenu();
11     CMenu(std::string, CMenuItem *, size_t);
12     int getSelect() const;
13     bool getRunning();
14     void setRunning(bool);
15     std::string getTitle();
16     size_t getCount() const;
17     CMenuItem *getItems();
18     void print();
19     void printTitle();
20     int runCommand();
21
22 private:
23     int m_select{-1};
24     size_t m_count{};
25     bool m_running{true};
26     std::string m_title{};
27     CMenuItem *m_items{};
28 };
29
30 #endif // MYMENU_CMENU_H

```

## Cmenu.cpp

```

1  #include "CMenu.h"
2  #include "MyError.h"
3  #include "Tools.h"
4  #include <iostream>
5
6  namespace ExpressionTree
7  {
8      CMenu::CMenu() {}
9      CMenu::CMenu(std::string title, CMenuItem *items, size_t count) : m_title(title),
m_items(items), m_count(count) {}
10
11     int CMenu::getSelect() const
12     {
13         return m_select;
14     }
15
16     bool CMenu::getRunning()
17     {
18         return m_running;
19     }
20     void CMenu::setRunning(bool _running)
21     {
22         m_running = _running;
23     }
24
25     size_t CMenu::getCount() const
26     {
27         return m_count;
28     }
29
30     std::string CMenu::getTitle()
31     {
32         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();

```

```

103         }
104     }
105 }
106 }

```

## CmenuItem.h

```

1  #ifndef MYMENU_CPP_CMENUIITEM_H
2  #define MYMENU_CPP_CMENUIITEM_H
3  #include <string>
4  #include <functional>
5
6  using Func = std::function<int()>;
7
8  class CMenuItem
9  {
10 public:
11     CMenuItem(std::string, Func);
12     Func m_func{};
13     std::string m_item_name{};
14     std::string getName();
15     void print();
16     int run();
17 };
18
19 #endif // MYMENU_CPP_CMENUIITEM_H

```

## CmenuItem.cpp

```

1  #include "CMenuItem.h"
2  #include <iostream>
3
4  CMenuItem::CMenuItem(std::string item_name, Func func) : m_item_name(item_name), m_func(func) {}
5
6  std::string CMenuItem::getName()
7  {
8      return m_item_name;
9  }
10
11 void CMenuItem::print()
12 {
13     std::cout << m_item_name;
14 }
15
16 int CMenuItem::run()
17 {
18     return m_func();
19 }

```

## BinTree.h

```

1  #ifndef BINTREE_H
2  #define BINTREE_H
3
4  #include <fstream>
5  #include <iostream>
6  #include <queue>
7
8  namespace ExpressionTree
9  {
10
11     struct Node
12     {

```

```

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    {

```

```

17         m_error = error;
18         logging();
19     }
20
21     void MyError::logging()
22     {
23         std::fstream file;
24         auto now = std::chrono::system_clock::now();
25         std::time_t end_time = std::chrono::system_clock::to_time_t(now);
26         file.open(m_file, std::ios::app);
27         file << "WARNING: " << m_error.c_str() << "|" << std::ctime(&end_time);
28         file.close();
29     }
30 }

```

## MyError.h

```

1  #ifndef MYERROR_H
2  #define MYERROR_H
3  #include <string>
4  #include <string_view>
5
6  namespace ExpressionTree
7  {
8      class MyError
9      {
10     public:
11         MyError(std::string error);
12         static std::string m_file;
13         const char* getError() const;
14
15     private:
16         void logging();
17         std::string m_error;
18
19
20     };
21 }
22 #endif // MYERROR_H

```

## Tools.cpp

```

1  #include <iostream>
2  #include <limits>
3  #include "Tools.h"
4
5
6
7  #ifdef _WIN32
8  #include <windows.h>
9  #endif
10
11 void ClearScreen() {
12     #ifdef _WIN32
13         // Очистка экрана для Windows
14         system("cls");
15     #else
16         // Очистка экрана для Linux/Unix
17         system("clear");
18     #endif
19 }
20
21 void WaitForEnter()
22 {
23     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.