|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **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. ИУК4-42Б | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( Карельский М.К. )  (Подпись) |
| Проверил: | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( Пчелинцева Н.И. )  (Подпись) |
| Дата сдачи (защиты):  Результаты сдачи (защиты): | | |
|  | - Балльная оценка:  - Оценка: | |

Калуга, 2022

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

**Задание:**

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 7**

Построить бинарное дерево следующего выражения 9 + (8 \* (7 + (6 \* (5 + (4 \* 3))))) и вывести его на экран. Написать процедуры постфиксного, инфиксного и префиксного обхода дерева и вывести соответствующие выражения.

**Листинг:**

***ExpressionTree.h***

#pragma once

#include <set>

#include <vector>

#include <string>

#include <fstream>

namespace DTAS

{

class ExpressionTree

{

public:

ExpressionTree();

~ExpressionTree();

bool IsEmpty();

void Create();

void Create(std::ifstream& stream);

void Print();

void Change();

void Insert();

void Remove();

void Clear();

void BypassDirectly();

void BypassSymmetrical();

void BypassConversely();

std::string GetExpressionDirectly();

std::string GetExpressionSymmetrical();

std::string GetExpressionConversely();

private:

struct Number

{

int number{};

Number\* left{};

Number\* right{};

};

enum Selection

{

None,

Left,

Right

};

void Print(Number\* elementToHighlight,

Selection selection = None);

void Place(Number\* pointer, std::vector<int> branch);

void Place(Number\* pointer, std::ifstream& stream);

void DrawElement(Number\* pointer,

std::set<unsigned short> columnsToDraw,

unsigned short numberPosition,

Number\* elementToHighlight = nullptr,

Selection selection = None);

unsigned short Select(Number\* pointer,

Number\* &selected);

Number\* GoUp(Number\* target, Number\* current);

void Clear(Number\* target);

void BypassDirectly(Number\* number);

void BypassSymmetrical(Number\* number);

void BypassConversely(Number\* number);

void AddMemberDirectly(Number\* pointer, std::string& destination);

void AddMemberSymmetrical(Number\* pointer, std::string& destination);

void AddMemberConversely(Number\* pointer, std::string& destination);

Number\* \_root{};

};

}

***ExpressionTree.cpp***

#include "ExpressionTree.h"

#include <iostream>

namespace DTAS

{

ExpressionTree::ExpressionTree() : \_root(nullptr) {}

ExpressionTree::~ExpressionTree() { Clear(); }

bool ExpressionTree::IsEmpty() { return \_root == nullptr; }

void ExpressionTree::Create()

{

std::cout << "Input root: ";

int input{};

std::cin >> input;

\_root = new Number{ input, nullptr, nullptr };

std::vector<int> branch{};

branch.push\_back(input);

Place(\_root, branch);

}

void ExpressionTree::Place(Number\* pointer, std::vector<int> branch)

{

int input{};

std::string branchStr{};

for (size\_t i{}; i < branch.size() - 1; ++i)

{

branchStr += std::to\_string(branch[i]);

branchStr += " - ";

}

branchStr += std::to\_string(branch[branch.size() - 1]);

branchStr += "\n";

system("cls");

Print(pointer, Left);

std::cout << "\nBranch: " << branchStr << "\n";

std::cout << "Place left? (1 = yes)\n";

std::cout << ">>> ";

std::cin >> input;

if (input == 1)

{

std::cout << "Input number: ";

std::cin >> input;

pointer->left = new Number{ input, nullptr, nullptr };

branch.push\_back(input);

Place(pointer->left, branch);

branch.pop\_back();

}

system("cls");

Print(pointer, Right);

std::cout << "\nBranch: " << branchStr << "\n";

std::cout << "Place right? (1 = yes)\n";

std::cout << ">>> ";

std::cin >> input;

if (input == 1)

{

std::cout << "Input number: ";

std::cin >> input;

pointer->right = new Number{ input, nullptr, nullptr };

branch.push\_back(input);

Place(pointer->right, branch);

branch.pop\_back();

}

}

void ExpressionTree::Create(std::ifstream& stream)

{

int number{};

stream >> number;

\_root = new Number{ number, nullptr, nullptr };

Place(\_root, stream);

}

void ExpressionTree::Place(Number\* pointer, std::ifstream& stream)

{

int input{};

stream >> input;

if (input == 1)

{

stream >> input;

pointer->left = new Number{ input, nullptr, nullptr };

Place(pointer->left, stream);

}

stream >> input;

if (input == 1)

{

stream >> input;

pointer->right = new Number{ input, nullptr, nullptr };

Place(pointer->right, stream);

}

}

void ExpressionTree::Print()

{

if (!IsEmpty())

{

std::set<unsigned short> columnsToDraw{};

DrawElement(\_root, columnsToDraw, 0);

}

else

std::cout << "The tree is empty\n";

}

void ExpressionTree::Print(Number\* elementToHighlight, Selection selection)

{

if (!IsEmpty())

{

std::set<unsigned short> columnsToDraw{};

DrawElement(\_root, columnsToDraw, 0, elementToHighlight, selection);

}

}

void ExpressionTree::DrawElement(Number\* pointer,

std::set<unsigned short> columnsToDraw, unsigned short numberPosition,

Number\* elementToHighlight, Selection selection)

{

std::cout << pointer->number;

if (pointer == elementToHighlight)

std::cout << " <=====";

std::cout << "\n";

for (unsigned short i{}; i < numberPosition; ++i)

if (columnsToDraw.find(i) != columnsToDraw.end())

std::cout << "|";

else

std::cout << " ";

std::cout << "|";

if (pointer->left != nullptr)

{

std::cout << ">";

columnsToDraw.insert(numberPosition);

DrawElement(pointer->left, columnsToDraw,

numberPosition + 2, elementToHighlight, selection);

columnsToDraw.erase(numberPosition);

}

else

if (pointer == elementToHighlight && selection == Left)

std::cout << "#\n";

else

std::cout << ">\n";

for (unsigned short i{}; i < numberPosition; ++i)

if (columnsToDraw.find(i) != columnsToDraw.end())

std::cout << "|";

else

std::cout << " ";

std::cout << "|";

if (pointer->right != nullptr)

{

std::cout << ">";

DrawElement(pointer->right, columnsToDraw,

numberPosition + 2, elementToHighlight, selection);

}

else

if (pointer == elementToHighlight && selection == Right)

std::cout << "#\n";

else

std::cout << ">\n";

}

unsigned short ExpressionTree::Select(Number\* pointer, Number\*& selected)

{

unsigned short code = 1;

while (code != 0 && code != 4)

{

system("cls");

Print(pointer);

std::cout << "\n1. Move left\n";

std::cout << "2. Move right\n";

std::cout << "3. Return\n";

std::cout << "4. Select\n";

std::cout << "0. Cancel\n";

std::cout << ">>> ";

std::cin >> code;

if (code == 1)

{

if (pointer->left != nullptr)

code = Select(pointer->left, selected);

}

else if (code == 2)

{

if (pointer->right != nullptr)

code = Select(pointer->right, selected);

}

else if (code == 3)

{

if (pointer != \_root)

return 3;

}

else if (code == 4)

{

selected = pointer;

return 4;

}

else if (code == 0)

{

return 0;

}

}

}

void ExpressionTree::Change()

{

if (!IsEmpty())

{

Number\* selected{};

if (Select(\_root, selected) == 4)

{

std::cout << "\nInput number: ";

int number{};

std::cin >> number;

selected->number = number;

}

}

}

void ExpressionTree::Insert()

{

if (!IsEmpty())

{

Number\* selected{};

if (Select(\_root, selected) == 4)

{

std::cout << "\n1. Place subtree left\n";

std::cout << "2. Place subtree right\n";

std::cout << ">>> ";

unsigned short placement{};

std::cin >> placement;

std::cout << "\nInput number: ";

int number{};

std::cin >> number;

Number\* moved = new Number{selected->number,

selected->left, selected->right};

selected->number = number;

if (placement == 1)

{

selected->left = moved;

selected->right = nullptr;

}

else if (placement == 2)

{

selected->left = nullptr;

selected->right = moved;

}

}

}

}

void ExpressionTree::Remove()

{

if (!IsEmpty())

{

Number\* selected{};

if (Select(\_root, selected) == 4)

{

Number\* parent = GoUp(selected, \_root);

if (selected->left == nullptr && selected->right == nullptr)

{

if (parent == nullptr)

{

delete \_root;

\_root = nullptr;

}

else

{

if (parent->left == selected)

{

delete parent->left;

parent->left = nullptr;

}

else

{

delete parent->right;

parent->right = nullptr;

}

}

}

else if (selected->left != nullptr && selected->right != nullptr)

{

Number\* numberToSwap = selected->left;

Number\* numberToSwapParent = selected;

while (numberToSwap->right != nullptr)

{

numberToSwapParent = numberToSwap;

numberToSwap = numberToSwap->right;

}

numberToSwap->right = selected->right;

if (numberToSwapParent != selected)

{

numberToSwap->left = selected->left;

if (numberToSwapParent->left == numberToSwap)

numberToSwapParent->left = nullptr;

else

numberToSwapParent->right = nullptr;

}

if (parent != nullptr)

{

if (parent->left == selected)

{

delete parent->left;

parent->left = numberToSwap;

}

else

{

delete parent->right;

parent->right = numberToSwap;

}

}

else

{

delete \_root;

\_root = numberToSwap;

}

}

else

{

if (parent == nullptr)

{

if (\_root->left != nullptr)

{

Number\* temp = \_root->left;

delete \_root;

\_root = temp;

}

else

{

Number\* temp = \_root->right;

delete \_root;

\_root = temp;

}

}

else

{

Number\* numberToSet{};

if (selected->left != nullptr)

numberToSet = selected->left;

else

numberToSet = selected->right;

if (parent->left == selected)

{

delete parent->left;

parent->left = numberToSet;

}

else

{

delete parent->right;

parent->right = numberToSet;

}

}

}

}

}

}

ExpressionTree::Number\* ExpressionTree::GoUp(Number\* target, Number\* current)

{

if (current->left == target || current->right == target)

return current;

Number\* numberToFind = nullptr;

if (current->left != nullptr)

numberToFind = GoUp(target, current->left);

if (numberToFind == nullptr && current->right != nullptr)

numberToFind = GoUp(target, current->right);

return numberToFind;

}

void ExpressionTree::Clear()

{

if (!IsEmpty())

{

Clear(\_root);

\_root = nullptr;

}

}

void ExpressionTree::Clear(Number\* target)

{

if (target->left != nullptr)

Clear(target->left);

if (target->right != nullptr)

Clear(target->right);

delete target;

}

void ExpressionTree::BypassDirectly()

{

if (!IsEmpty())

BypassDirectly(\_root);

}

void ExpressionTree::BypassDirectly(Number\* number)

{

std::cout << number->number << " ";

if (number->left != nullptr)

BypassDirectly(number->left);

if (number->right != nullptr)

BypassDirectly(number->right);

}

void ExpressionTree::BypassSymmetrical()

{

if (!IsEmpty())

BypassSymmetrical(\_root);

}

void ExpressionTree::BypassSymmetrical(Number\* number)

{

if (number->left != nullptr)

BypassSymmetrical(number->left);

std::cout << number->number << " ";

if (number->right != nullptr)

BypassSymmetrical(number->right);

}

void ExpressionTree::BypassConversely()

{

if (!IsEmpty())

BypassConversely(\_root);

}

void ExpressionTree::BypassConversely(Number\* number)

{

if (number->left != nullptr)

BypassConversely(number->left);

if (number->right != nullptr)

BypassConversely(number->right);

std::cout << number->number << " ";

}

std::string ExpressionTree::GetExpressionDirectly()

{

if (!IsEmpty())

{

std::string result = std::to\_string(\_root->number);

if (\_root->left != nullptr || \_root->right != nullptr)

{

result += " + ";

if (\_root->left != nullptr)

AddMemberDirectly(\_root->left, result);

if (\_root->left != nullptr && \_root->right != nullptr)

result += " \* ";

if (\_root->right != nullptr)

AddMemberDirectly(\_root->right, result);

}

return result;

}

return "";

}

void ExpressionTree::AddMemberDirectly(Number\* pointer, std::string& destination)

{

if (pointer->left != nullptr || pointer->right != nullptr)

{

destination += "(";

destination += std::to\_string(pointer->number);

destination += " + ";

if (pointer->left != nullptr)

AddMemberDirectly(pointer->left, destination);

if (pointer->left != nullptr && pointer->right != nullptr)

destination += " \* ";

if (pointer->right != nullptr)

AddMemberDirectly(pointer->right, destination);

destination += ")";

}

else

destination += std::to\_string(pointer->number);

}

std::string ExpressionTree::GetExpressionSymmetrical()

{

if (!IsEmpty())

{

std::string result = "";

AddMemberSymmetrical(\_root, result);

return result;

}

return "";

}

void ExpressionTree::AddMemberSymmetrical(Number\* pointer, std::string& destination)

{

if (pointer->left != nullptr)

{

AddMemberSymmetrical(pointer->left, destination);

destination += " + ";

}

destination += std::to\_string(pointer->number);

if (pointer->right != nullptr)

{

destination += " \* ";

if (pointer->right->left != nullptr)

{

destination += "(";

AddMemberSymmetrical(pointer->right, destination);

destination += ")";

}

else

AddMemberSymmetrical(pointer->right, destination);

}

}

std::string ExpressionTree::GetExpressionConversely()

{

if (!IsEmpty())

{

std::string result = "";

AddMemberConversely(\_root, result);

return result;

}

return "";

}

void ExpressionTree::AddMemberConversely(Number\* pointer, std::string& destination)

{

if (pointer->left != nullptr || pointer->right != nullptr)

{

if (pointer->left != nullptr)

{

if ((pointer->left->left != nullptr || pointer->left->right != nullptr)

&& pointer->right != nullptr)

{

destination += "(";

AddMemberConversely(pointer->left, destination);

destination += ")";

}

else

AddMemberConversely(pointer->left, destination);

}

if (pointer->left != nullptr && pointer->right != nullptr)

destination += " \* ";

if (pointer->right != nullptr)

{

if ((pointer->right->left != nullptr || pointer->right->right != nullptr)

&& pointer->left != nullptr)

{

destination += "(";

AddMemberConversely(pointer->right, destination);

destination += ")";

}

else

AddMemberConversely(pointer->right, destination);

}

destination += " + ";

}

destination += std::to\_string(pointer->number);

}

}

***Menu.h***

#pragma once

#include "ExpressionTree.h"

namespace DTAS

{

class Menu

{

public:

Menu();

Menu(std::string input,

std::string output = "Output.txt",

std::string logFile = "Log.txt");

void Run();

private:

void Pause();

std::string \_output{};

std::string \_logFile{};

ExpressionTree \_tree{};

};

}

***Menu.cpp***

#include "Menu.h"

#include <fstream>

#include <iostream>

namespace DTAS

{

Menu::Menu() : \_output("Output.txt"), \_logFile("Log.txt"), \_tree() {}

Menu::Menu(std::string input, std::string output,

std::string logFile) : \_output(output), \_logFile(logFile)

{

std::ifstream inputData(input);

if (inputData)

\_tree.Create(inputData);

else

{

std::ofstream errorData(\_logFile, std::ios\_base::app);

errorData << "Input file wasn't found\n";

errorData.close();

std::cout << "Input file wasn't found";

Pause();

}

inputData.close();

}

void Menu::Run()

{

unsigned short input = 1;

while (input != 0)

{

system("cls");

\_tree.Print();

std::cout << "\n1. Create tree\n";

std::cout << "2. Delete tree\n\n";

std::cout << "3. Change element\n";

std::cout << "4. Insert element\n";

std::cout << "5. Remove element\n\n";

std::cout << "6. Bypass directly\n";

std::cout << "7. Bypass symmetrical\n";

std::cout << "8. Bypass conversely\n\n";

std::cout << "9. Form expression directly\n";

std::cout << "10. Form expression symmetrical\n";

std::cout << "11. Form expression conversely\n\n";

std::cout << "0. Exit\n";

std::cout << ">>> ";

std::cin >> input;

std::cout << "\n";

std::string result{};

switch (input)

{

case 1:

system("cls");

\_tree.Clear();

\_tree.Create();

break;

case 2:

\_tree.Clear();

break;

case 3:

system("cls");

\_tree.Change();

break;

case 4:

system("cls");

\_tree.Insert();

break;

case 5:

system("cls");

\_tree.Remove();

break;

case 6:

\_tree.BypassDirectly();

if (!\_tree.IsEmpty())

Pause();

break;

case 7:

\_tree.BypassSymmetrical();

if (!\_tree.IsEmpty())

Pause();

break;

case 8:

\_tree.BypassConversely();

if (!\_tree.IsEmpty())

Pause();

break;

case 9:

result = \_tree.GetExpressionDirectly();

std::cout << result;

if (!\_tree.IsEmpty())

{

std::ofstream outputData(\_output);

outputData << result;

outputData.close();

Pause();

}

break;

case 10:

result = \_tree.GetExpressionSymmetrical();

std::cout << result;

if (!\_tree.IsEmpty())

{

std::ofstream outputData(\_output);

outputData << result;

outputData.close();

Pause();

}

break;

case 11:

result = \_tree.GetExpressionConversely();

std::cout << result;

if (!\_tree.IsEmpty())

{

std::ofstream outputData(\_output);

outputData << result;

outputData.close();

Pause();

}

break;

}

}

}

void Menu::Pause()

{

system("pause>pauseTemp");

remove("pauseTemp");

}

}

***Main.cpp***

#include "Menu.h"

int main(int argc, char\* argv[])

{

if (argc == 1)

{

DTAS::Menu menu{};

menu.Run();

}

else if (argc == 2)

{

DTAS::Menu menu{ argv[1] };

menu.Run();

}

else if (argc == 3)

{

DTAS::Menu menu{ argv[1], argv[2] };

menu.Run();

}

else

{

DTAS::Menu menu{ argv[1], argv[2], argv[3] };

menu.Run();

}

return 0;

}

**Результат:**

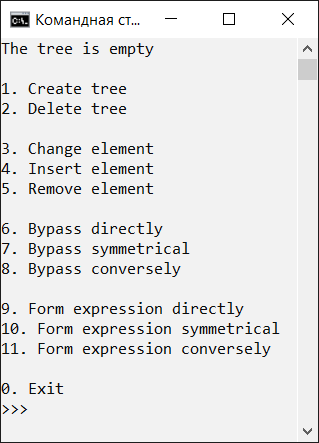
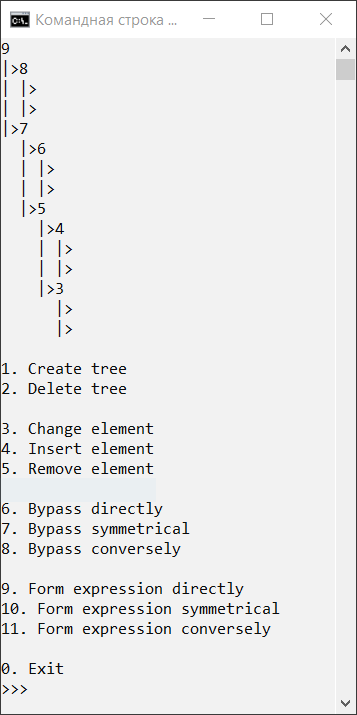
 

Рис. 1. Дерево (пустое и заполненное) и основное меню

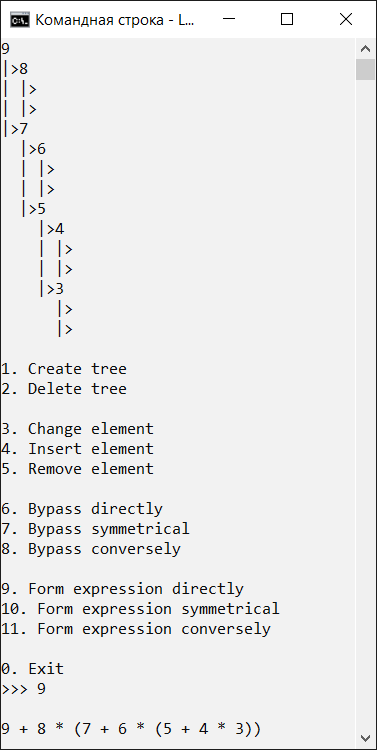
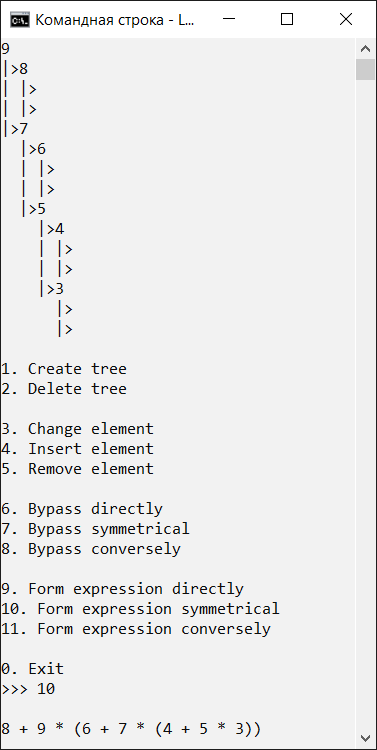
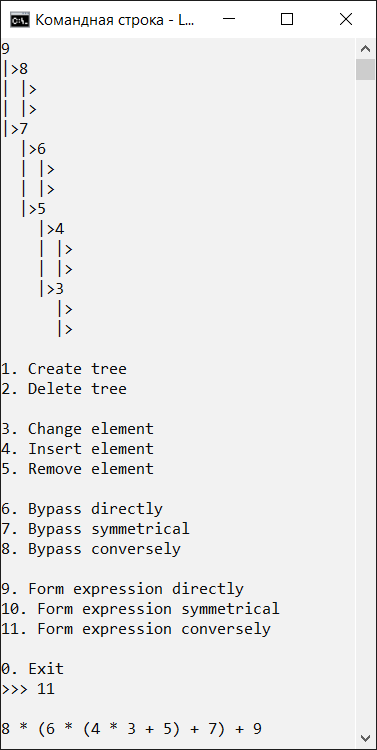
  

Рис. 2. Сформированные выражения

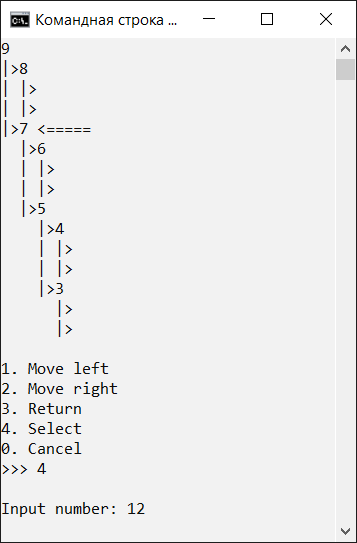
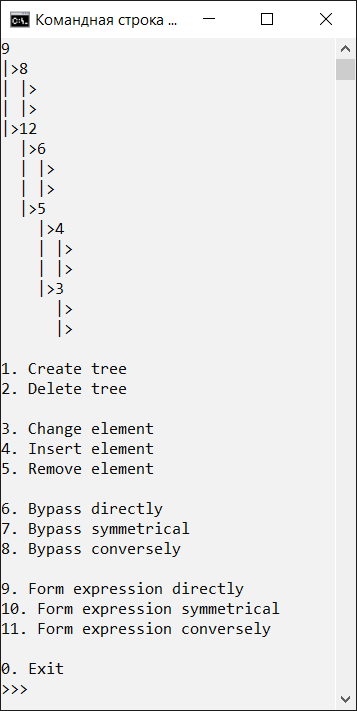
 

Рис. 3. Изменение элемента

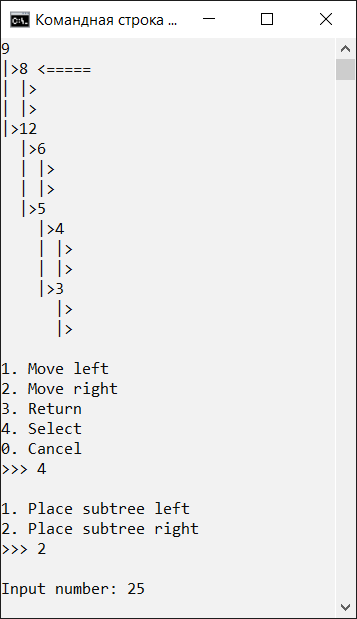
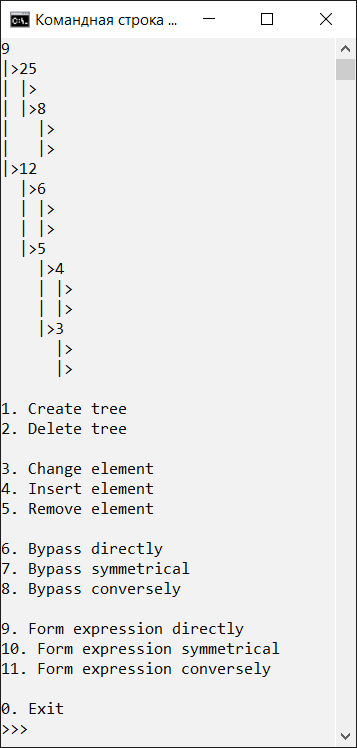
 

Рис. 4. Вставка элемента

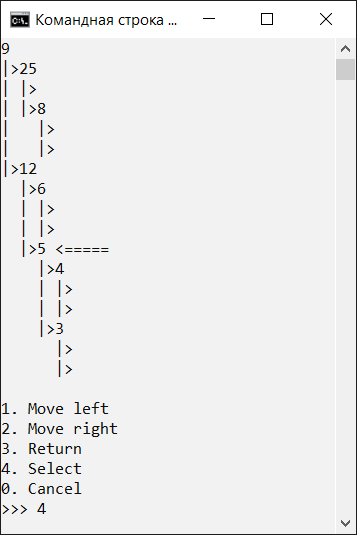
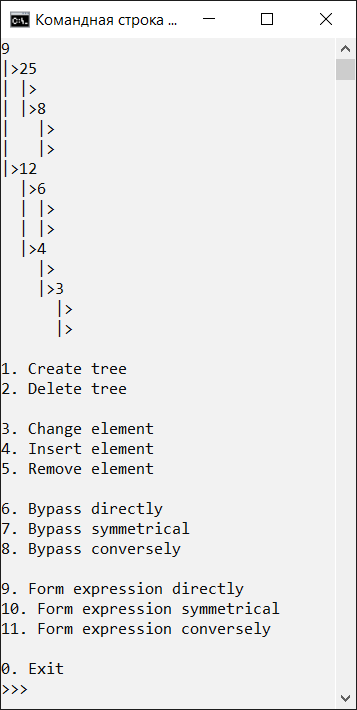
 

Рис. 5. Удаление элемента

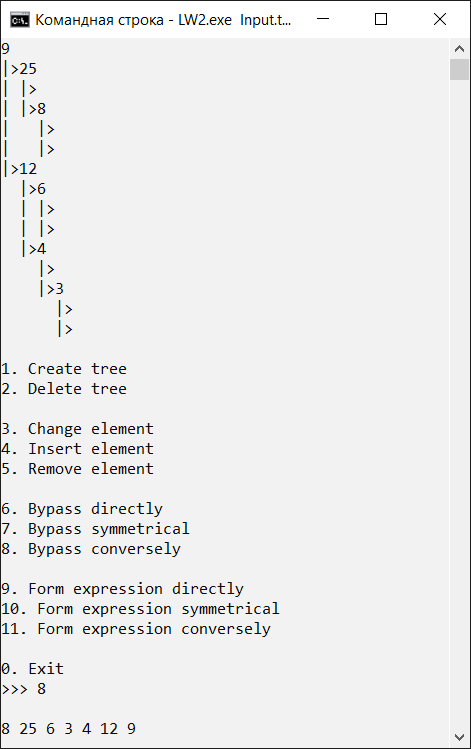
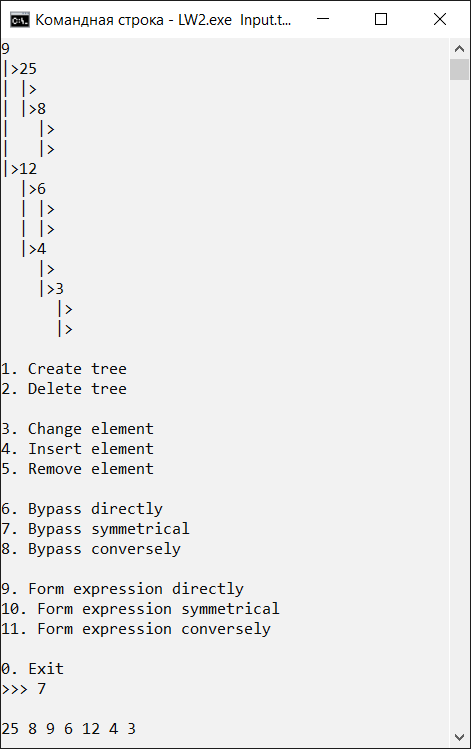
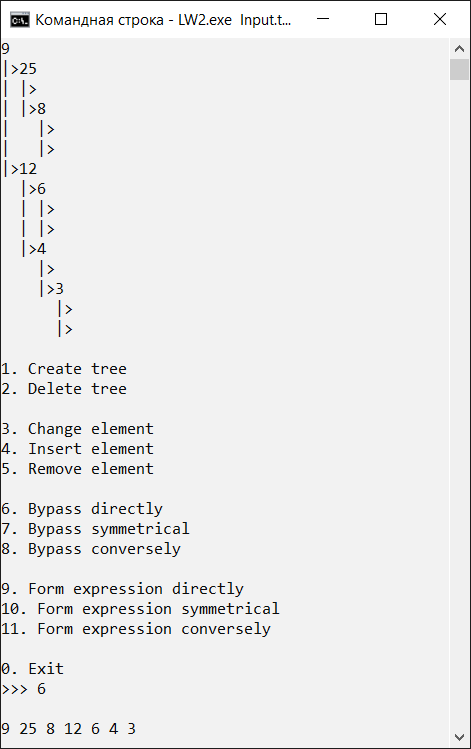


Рис. 6. Обходы

**Вывод:** в ходе выполнения лабораторной работы были получены навыки создания бинарного дерева, реализации его заполнения и удаления, проверки дерева на пустоту, вывода его в консоль, изменения и удаления его элемента, вставки в него нового элемента, совершения прямого, симметричного и обратного обходов.