Изображение выглядит как текст, керамическая посуда

Автоматически созданное описание

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**"МИРЭА - Российский технологический университет"**

**РТУ МИРЭА**

Институт информационных технологий (ИТ)

Кафедра математического обеспечения и стандартизации информационных технологий (МОСИТ)

**ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №4**

**по дисциплине**

**«Структуры и алгоритмы обработки данных»**

Тема. Нелинейные структуры данных. Бинарное дерево.

|  |  |  |
| --- | --- | --- |
| Выполнил студент группы ИКБО-04-22 |  | Кликушин В.И. |
| Принял старший преподаватель |  | Скворцова Л.А. |

Москва 2023

ОГЛАВЛЕНИЕ

[1 СВОЙСТВА ДЕРЕВА 3](#_Toc150359886)

[2 ПРИМЕРЫ ДЕРЕВА 4](#_Toc150359887)

[3 ТЕСТИРОВАНИЕ ПРОГРАММЫ 7](#_Toc150359888)

[4 ОПИСАНИЕ ФУНКЦИОНАЛА ПРИЛОЖЕНИЯ 13](#_Toc150359889)

[5 КОД ОСНОВНОЙ ПРОГРАММЫ 17](#_Toc150359890)

[5.1 Код файла ExpressionTree.h: 17](#_Toc150359891)

[5.2 Код файла ExpressionTree.cpp: 18](#_Toc150359892)

[5.3 Код файла main.cpp: 23](#_Toc150359893)

[6 ОТВЕТЫ НА КОНТРОЛЬНЫЕ ВОПРОСЫ 26](#_Toc150359894)

[7 ВЫВОДЫ 29](#_Toc150359895)

[8 ИСПОЛЬЗУЕМЫЕ ИСТОЧНИКИ 30](#_Toc150359896)

# 1 СВОЙСТВА ДЕРЕВА

Выражение, содержащее бинарные операции, можно представить как бинарное дерево, в котором внутренние узлы будут хранить операции, а листья операнды. Если операнды – числа, то дерево можно использовать для вычисления значения выражения. Если операнды - переменные, то в узел операнда записывается ссылка на ячейки переменных. При этом необходимо учитывать приоритетность операций и правило: сначала надо вычислить значение выражения левого поддерева, затем правого, а затем выполнить операцию, находящуюся в корне, т.е., обходя дерево обратным обходом. Нисходящему обходу соответствует префиксная форма представления, т.к. в ней знак операции предшествует операнду. Восходящему обходу соответствует постфиксная форма представления, т.к. в ней знак операции находится после операндов. Смешанному обходу соответствует инфиксная форма представления, т.к. в ней знак операции находится между операндами. Для того чтобы задать приоритеты операций, используется абсолютно скобочная форма, в которой каждое подвыражение заключается в круглые скобки.

# 2 ПРИМЕРЫ ДЕРЕВА

Изобразим несколько деревьев выражений, которые будут являться тестами для тестирования функционала приложения.

1. Выражение 3+((5+9)\*2) будет соответствовать следующему дереву:

Изображение выглядит как диаграмма, зарисовка, рисунок, круг

Автоматически созданное описание

Рисунок 1 – Дерево выражений №1

1. Выражение 1\*2+3\*4 будет соответствовать следующему дереву:

Изображение выглядит как диаграмма, круг, линия, зарисовка

Автоматически созданное описание

Рисунок 2 – Дерево выражений №2

1. Выражение ((4/2)+(5\*7-(9+3)))/5 будет соответствовать следующему дереву:

Изображение выглядит как диаграмма, линия, круг, шаблон

Автоматически созданное описание

Рисунок 3 – Дерево выражений №3

# 3 ТЕСТИРОВАНИЕ ПРОГРАММЫ

1. Тест №1

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рисунок 4 – Создание и вывод дерева из теста №1

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 5 – Префиксная форма выражения дерева из теста №1

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 6 – Вычисление значения выражения по дереву из теста №1

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 7 – Вычисление значения выражения по префиксной форме из теста №1

1. Тест №2

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 8 - Создание и вывод дерева из теста №2

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 9 - Префиксная форма выражения дерева из теста №2

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 10 - Вычисление значения выражения по дереву из теста №2

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 11 - Вычисление значения выражения по префиксной форме из теста №2

1. Тест №3

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рисунок 12 - Создание и вывод дерева из теста №3

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 13 - Префиксная форма выражения дерева из теста №3

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 14 - Вычисление значения выражения по дереву из теста №3

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 15 - Вычисление значения выражения по префиксной форме из теста №3

# 4 ОПИСАНИЕ ФУНКЦИОНАЛА ПРИЛОЖЕНИЯ

Операции

1. Построение дерева выражения

Предусловие. Выражение, записанное в строку.

Постусловие. Нет возвращаемого значения. Строит дерево на основе введенного выражения.

**void createTree(string str);**

1. Сравнение приоритетов двух операций

Предусловие. Два параметра типа char - операторы.

Постусловие. Возвращает логическую единицу, если приоритет первого оператора выше, чем приоритет второго, иначе логический ноль.

**bool hasHigherPrecedence(char op1, char op2);**

1. Связывание узлов

Предусловие. Стек из указателей на объекты структуры узла дерева Node, передаваемый по ссылке, стек из символьных операторов, передаваемый по ссылке.

Постусловие. Нет возвращаемого значения.

**void applyOperator(stack<Node\*>& nodes, stack<char>& operators);**

1. Образование префиксной записи по дереву

Предусловие. Указатель на объект структуры данных Node– узел дерева.

Постусловие. Строковое значение, соответствующее префиксной записи выражения в дереве.

**string prefix(Node\* node);**

1. Образование инфиксной записи по дереву

Предусловие. Указатель на объект структуры данных Node– узел дерева.

Постусловие. Строковое значение, соответствующее инфиксной записи выражения в дереве.

**string getInfix(Node\* node);**

1. Подсчет значения выражения по дереву.

Предусловие. Указатель на объект структуры данных Node– узел дерева.

Постусловие. Значение выражения типа double.

**double evaluate(Node\* node);**

1. Подсчет значения выражения по префиксной форме.

Предусловие. Строковое значение – префиксная форма выражения.

Постусловие. Значение выражения типа double.

**double evaluatePrefixExpression(string prefix);**

1. Вывод дерева в консоль

Предусловие. Указатель на объект структуры данных Node– узел дерева, целочисленная переменная level, отвечающая за отступы.

Постусловие. Нет возвращаемого значения.

**void printTree(Node\* node, int level = 0);**

Опишем принцип работы некоторых функций.

1. Построение дерева.

Метод createTree создает дерево выражения на основе строки str. Внутри метода используются две стека - один для узлов дерева и один для операторов. Метод перебирает каждый символ в строке.  
- Если символ является цифрой, создается узел с этим символом в качестве данных и помещается в стек узлов.  
- Если символ является оператором (+, -, \*, /), то из стека операторов извлекаются все операторы с большим или равным приоритетом и применяются к двум верхним узлам из стека узлов. Результат операции помещается в новый узел, который затем помещается в стек узлов. Текущий оператор также помещается в стек операторов.  
- Если символ является открывающей скобкой, он просто помещается в стек операторов.  
- Если символ является закрывающей скобкой, из стека операторов извлекаются и применяются все операторы до открывающей скобки, а открывающая скобка удаляется из стека.

1. Связывание узлов.

Метод applyOperator применяет оператор к двум верхним узлам из стека узлов. Создается новый узел, в котором оператор становится данными, а два верхних узла становятся его потомками. Новый узел помещается в стек узлов.

1. Получение префиксной формы

Метод prefix работает рекурсивно для получения префиксной формы выражения из заданного узла node. Создаем переменную result для хранения текущего префиксного выражения и инициализируем ее пустой строкой. Проверяем, является ли заданный узел node нулевым указателем. Если да, то возвращаем пустую строку. Добавляем значение data текущего узла node к результату result. Это добавляет текущий оператор или операнд в префиксное выражение. Проверяем, есть ли у узла node левый или правый дочерний узел. Если есть, вызываем рекурсивно метод prefix для каждого из них и добавляем результаты в конец строки result. Это добавляет префиксные выражения левого и правого поддеревьев к текущему префиксному выражению.

1. Подсчет значения выражения по дереву.

Метод evaluate() принимает указатель на узел дерева выражения и рекурсивно вычисляет его значение. Проверяем, является ли переданный узел nullptr. Если да, возвращаем значение бесконечности, чтобы обработать ошибку. Проверяем, является ли значение узла цифрой. Если да, возвращаем цифру, преобразованную из символа в числовое значение. Если узел не является цифрой, рекурсивно вызываем evaluate() для его левого и правого поддеревьев, сохраняя результаты в переменные leftVal и rightVal соответственно. Используя значение узла, который является операцией, выполняем соответствующую операцию с leftVal и rightVal и возвращаем результат.

1. Подсчет значения выражения по префиксной записи.

Метод evaluatePrefixExpression(string prefix): вычисляет значение выражения в префиксной нотации. Используется стек операндов для хранения промежуточных результатов вычислений. Происходит итерация по каждому символу в строке префиксного выражения.  
- Если символ является цифрой, он преобразуется в целое число и помещается в стек операндов.  
- Если символ является оператором, извлекаются два верхних операнда из стека, выполняется операция и результат помещается обратно в стек.  
- Если символ является пробелом или табуляцией, он игнорируется.  
- Если символ не является ни цифрой, ни оператором, выводится сообщение об ошибке.

1. Вывод дерева в консоль.

Алгоритм метода проходит по дереву в глубину сначала справа, затем выводит значение узла и, наконец, обходит левое поддерево каждого узла. Это позволяет вывести дерево в виде дерева, где узлы постепенно спускаются на новые уровни вниз и смещаются вправо.

# 5 КОД ОСНОВНОЙ ПРОГРАММЫ

## 5.1 Код файла ExpressionTree.h:

#include <iostream>

#include <string>

#include <stack>

#include <iomanip>

using namespace std;

class ExpressionTree

{

private:

struct Node // структура узла

{

char data; // данные

Node\* left; // указатель на левое поддерево

Node\* right; // указатель на правое поддерево

Node(char data)

{

this->data = data;

this->left = nullptr;

this->right = nullptr;

}

};

Node\* root; // указатель на корень дерева

void printTree(Node\* node, int level = 0) // метод вывода дерева

{

if (node != nullptr)

{

level += 10;

printTree(node->right, level);

cout << endl;

for (int i = 0; i < level; i++)

cout << " ";

cout << node->data << "\n";

printTree(node->left, level);

}

}

public:

ExpressionTree(); // конструктор

Node\* createNode(char data) // создание нового узла

{

return new Node(data);

}

void createTree(string str); // метод построения дерева

bool hasHigherPrecedence(char op1, char op2); // сравнение приоритетов операций

void applyOperator(stack<Node\*>& nodes, stack<char>& operators); // применяет оператор к двум верхним узлам из стека узлов.Создается новый узел, в котором оператор становится данными, а два верхних узла становятся его потомками.Новый узел помещается в стек узлов.

string getPrefix();

string prefix(Node\* node);

string getInfix();

string getInfix(Node\* node);

double evaluateExpression();

double evaluate(Node\* node);

void printTree();

double evaluatePrefixExpression(string prefix);

};

## 5.2 Код файла ExpressionTree.cpp:

#include "ExpressionTree.h"

ExpressionTree::ExpressionTree()

{

root = nullptr;

}

void ExpressionTree::createTree(string str)

{

stack<Node\*> nodes; // стек для узлов дерева

stack<char> operators; // стек для операторов

for (char ch : str)

{

if (isdigit(ch)) // текущий символ - цифра

{

nodes.push(createNode(ch)); // создается новый узел и помещается в стек узлов дерева

}

else if (ch == '+' || ch == '-' || ch == '\*' || ch == '/') // текущий символ - знак операции

{

while (!operators.empty() && hasHigherPrecedence(operators.top(), ch))

{

applyOperator(nodes, operators);

}

operators.push(ch);

}

else if (ch == '(')

{

operators.push(ch);

}

else if (ch == ')')

{

while (!operators.empty() && operators.top() != '(')

{

applyOperator(nodes, operators);

}

if (!operators.empty() && operators.top() == '(')

{

operators.pop(); // Удалить '('

}

}

}

while (!operators.empty())

{

applyOperator(nodes, operators);

}

if (!nodes.empty())

{

root = nodes.top();

}

}

bool ExpressionTree::hasHigherPrecedence(char op1, char op2)

{

return (op1 == '\*' || op1 == '/') && (op2 == '+' || op2 == '-'); // сравнивает приоритеты операций

}

void ExpressionTree::applyOperator(stack<Node\*>& nodes, stack<char>& operators)

{

char op = operators.top();

operators.pop();

if (nodes.size() < 2)

{

return; // Error handling

}

Node\* right = nodes.top();

nodes.pop();

Node\* left = nodes.top();

nodes.pop();

Node\* newNode = createNode(op);

newNode->left = left;

newNode->right = right;

nodes.push(newNode);

}

void ExpressionTree::printTree()

{

printTree(root, 0); // вывод дерева

}

string ExpressionTree::getPrefix()

{

return prefix(root); // префиксная форма выражения

}

string ExpressionTree::prefix(Node\* node)

{

string result;

if (node != nullptr)

{

result += node->data;

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

{

result += prefix(node->left);

result += prefix(node->right);

}

}

return result;

}

string ExpressionTree::getInfix()

{

return getInfix(root);

}

string ExpressionTree::getInfix(Node\* node)

{

if (node != nullptr)

{

string infix = "";

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

{

infix += "(";

infix += getInfix(node->left);

infix += node->data;

infix += getInfix(node->right);

infix += ")";

}

else

{

infix += node->data;

}

return infix;

}

return "";

}

double ExpressionTree::evaluateExpression()

{

return evaluate(root);

}

double ExpressionTree::evaluate(Node\* node)

{

if (node != nullptr)

{

if (isdigit(node->data))

{

return node->data - '0';

}

else

{

double leftVal = evaluate(node->left);

double rightVal = evaluate(node->right);

switch (node->data)

{

case '+':

return leftVal + rightVal;

case '-':

return leftVal - rightVal;

case '\*':

return leftVal \* rightVal;

case '/':

return leftVal / rightVal;

}

}

}

return INFINITY;

}

double ExpressionTree::evaluatePrefixExpression(string prefix)

{

stack<double> operands;

for (int i = prefix.length() - 1; i >= 0; i--)

{

if (isdigit(prefix[i]))

{

operands.push(prefix[i] - '0');

}

else if (prefix[i] == ' ' || prefix[i] == '\t')

{

continue;

}

else

{

double operand1 = operands.top();

operands.pop();

double operand2 = operands.top();

operands.pop();

switch (prefix[i])

{

case '+':

operands.push(operand1 + operand2);

break;

case '-':

operands.push(operand1 - operand2);

break;

case '\*':

operands.push(operand1 \* operand2);

break;

case '/':

operands.push(operand1 / operand2);

break;

default:

return INFINITY; // Error handling

}

}

}

if (operands.size() == 1)

{

return operands.top();

}

else

{

return INFINITY; // Error handling

}

}

## 5.3 Код файла main.cpp:

#include "ExpressionTree.h"

int main()

{

system("chcp 1251");

string expression,prefix,infix;

double res;

ExpressionTree tree;

int num;

while (true)

{

cout << "-----------------------------------------------------------------------------------------------------------" << endl;

cout << " Меню\n";

cout << " 1. Создать дерево выражений в соответствии с вводимым выражением\n";

cout << " 2. Отобразить дерево на экране\n";

cout << " 3. Образовать префиксную форму выражения, содержащегося в дереве\n";

cout << " 4. Вычислить значение выражения по дереву\n";

cout << " 5. Вычислить значение выражение по префиксной форме\n";

cout << " 6. Образовать инфиксную форму выражения, содержащегося в дереве\n";

cout << " 7. Завершить выполнение программы\n";

cin >> num;

switch (num)

{

case 1:

{

cout << " Введите выражение: ";

cin >> expression;

tree.createTree(expression);

break;

}

case 2:

{

cout << "Printed Tree:\n";

tree.printTree();

break;

}

case 3:

{

prefix = tree.getPrefix();

if (prefix.size() == 0)

cout << " Дерево выражения не создано" << endl;

else

cout << prefix << endl;

break;

}

case 4:

{

res = tree.evaluateExpression();

if (res == INFINITY)

cout << " Дерево выражения не создано или было вызвано исключение division by zero" << endl;

else

cout << res << endl;

break;

}

case 5:

{

prefix = tree.getPrefix();

if (prefix.size() == 0)

cout << " Дерево выражения не создано" << endl;

else

cout << tree.evaluatePrefixExpression(prefix) << endl;

break;

}

case 6:

{

infix = tree.getInfix();

if (infix.size() == 0)

cout << " Дерево выражения не создано" << endl;

else

cout << infix << endl;

break;

}

case 7:

return 0;

default:

{

cout << " Нет такого пункта\n";

}

}

}

}

# 6 ОТВЕТЫ НА КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что определяет степень дерева?

Степень дерева (арность дерева) - максимальная степень его узлов. Степень узла – количество дочерних узлов (непосредственных потомков).

1. Какова степень сильноветвящегося дерева?

К сильноветвящимся относятся деревья, степень которых больше двух. У k-арного дерева степень равна k.

1. Что определяет путь в дереве?

Путь в дереве к заданной вершине – последовательность (список) меток вершин дерева от корня к указанному узлу.

1. Как рассчитать длину пути в дереве?

Минимальное количество узлов от начального до конечного.

1. Какова степень бинарного дерева?

Не более двух.

1. Может ли дерево быть пустым?

Да, может.

1. Дайте определение бинарного дерева?

Бинарное (двоичное) дерево — это дерево, степень которого не больше двух.

1. Дайте определение алгоритму обхода.

Обход дерева – это алгоритм, обеспечивающий посещение каждого узла дерева с целью выполнить операцию с данными узла.

1. Приведите рекуррентную зависимость для вычисления высоты дерева.

Если текущий узел – нулевой указатель, вернуть ноль, иначе вернуть 1 + максимальное значение глубины от левого и правого поддерева

1. Изобразите бинарное дерево, корень которого имеет индекс 6, и которое представлено в памяти таблицей вида

Изображение выглядит как текст, число, Шрифт, чек

Автоматически созданное описание

Изображение выглядит как диаграмма, круг, линия

Автоматически созданное описание

1. Укажите путь обхода дерева по алгоритму: прямой; обратный; симметричный

Изображение выглядит как зарисовка, рисунок, диаграмма, круг

Автоматически созданное описание

Прямой обход: A,B,D,G,C,E,H,I,F

Обратный обход: G,D,B,H,I,E,F,C,A

Симметричный обход: D,G,B,A,H,E,I,C,F

1. Какая структура используется в алгоритме обхода дерева методом в «ширину»?

Очередь

1. Выведите путь при обходе дерева в «ширину». Продемонстрируйте использование структуры при обходе дерева.

Изображение выглядит как зарисовка, диаграмма, рисунок, круг

Автоматически созданное описание

Обход в ширину: A,B,C,D,E,F,G,H,I

1. Какая структура используется в не рекурсивном обходе дерева методом в «глубину»?

Стек

# 7 ВЫВОДЫ

В ходе практической работы были получены навыки работы с бинарными деревьями – деревом выражения.

# 8 ИСПОЛЬЗУЕМЫЕ ИСТОЧНИКИ

1. Учебно-методическое пособие СиАОД (часть 2)

2. Приложение к практическим работам – СДО (online-edu.mirea.ru)