МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Харківський національний університет імені В.Н.Каразіна

Факультет математики і інформатики

Кафедра теоретичної та прикладної інформатики

Індивідуальне завдання № 2

з курсу «Алгоритми і структури данних»

(назва дисципліни)

на тему: «Бінарні дерева»

Виконав: студент 2 курсу групи мф-21

напряму підготовки (спеціальності)

Комп'ютерні науки

122 Комп'ютерні науки

Єлагін І.А.

Прийняв: _____

Національна шкала: _____

Кількість балів: _____

Оцінка: ECTS _____

Харків – 2024

Завдання:

Формулу виду

<формула>::=<цифра>|(<формула><знак><формула>)

<знак>::= + | - | *

<цифра>::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

можна подати у вигляді двійкового дерева ("дерева-формули") згідно з наступними правилами: формула з однієї цифри представляється деревом з однієї вершини з цією цифрою, а формула виду (f1 s f2) – деревом, у якому корінь – це знак s, а ліве та праве піддерева – це відповідні подання формул f1 та f2. Приклад: (5*(3+8))

Код:

```
#include <iostream>
#include <string>

using namespace std;

// Структура вузла дерева
struct Node {
    string value;
    Node* left;
    Node* right;

    Node(string val = "") {
        value = val;
        left = nullptr;
        right = nullptr;
    }
};

// Рекурсивна функція побудови дерева
void buildTree(Node*& root, const string& formula, int& pos) {
    bool restart = true;
    while (restart){
        restart = false;
```

```
// Пропустити пробіли
while (pos < formula.length() && formula[pos] == ' ') {
pos++;
}

// Перевіряємо межу строки
if (pos >= formula.length()) return;

// Якщо поточний символ - число
if (isdigit(formula[pos])) {
   string num;
   while (pos < formula.length() && isdigit(formula[pos])) {
      num += formula[pos++];
   }

   Node* newNode = new Node(num);
   if (root->value.empty()) {
      root->left = newNode;
   }
   else {
      root->right = newNode;
   }

   restart = true;
   continue;
}

// Якщо поточний символ - оператор
if (formula[pos] == '+' || formula[pos] == '-' || formula[pos] == '*' || formula[pos] == '/') {
   root->value = formula[pos++];
   restart = true;
   continue;
}
```

```
        // Якщо поточний символ - відкриваюча дужка '('
        if (formula[pos] == '(') {
            pos++;
            Node* newNode = new Node();
            if (root->value.empty()) {
                root->left = newNode;
            }
            else {
                root->right = newNode;
            }
            buildTree(newNode, formula, pos);
            restart = true;
            continue;
        }

        // Якщо поточний символ - закриваюча дужка ')'
        if (formula[pos] == ')') {
            pos++;
            return;
        }
    }
}

// Допоміжна функція для тестування (обхід дерева)
void printTree(Node* root, int depth = 0) {
    if (!root) return;
    printTree(root->right, depth + 1);
    cout << string(depth * 4, ' ') << root->value << endl;
    printTree(root->left, depth + 1);
}

// Функція для обчислення значення виразу з дерева
double evaluateTree(Node* root) {
    if (!root) {
```

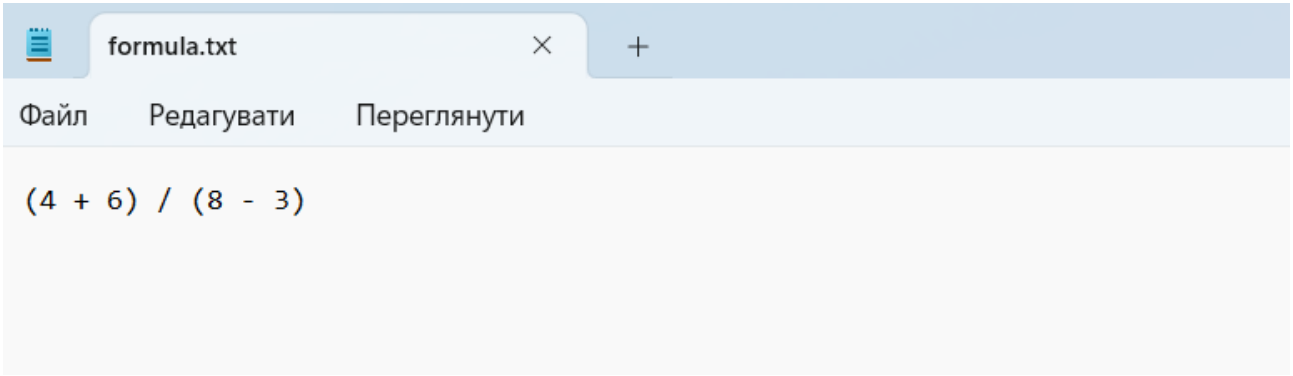
```
    cout << "Errore: Wrong tree." << endl; // Дерево не повинно бути порожнім
    return -999999999;
}

// Якщо це лист (число)
if (!root->left && !root->right) {
    return stod(root->value); // Конвертуємо рядок у число
}

// Рекурсивно обчислюємо ліву і праву частини дерева
double leftValue = evaluateTree(root->left);
double rightValue = evaluateTree(root->right);

// Виконуємо операцію залежно від значення вузла
if (root->value == "+") {
    return leftValue + rightValue;
}
else if (root->value == "-") {
    return leftValue - rightValue;
}
else if (root->value == "*") {
    return leftValue * rightValue;
}
else if (root->value == "/") {
    if (rightValue == 0) {
        cout << "Errore: Division by zero" << endl;
        return -999999999;
    }
    return leftValue / rightValue;
}
else {
    cout << " Unknown operator: " << root->value << endl; // Некоректний оператор
    return -999999999;
}
```

```
}

// Функція для друку дерева у префіксній формі (Polish notation)
void printPrefix(Node* root) {
    if (!root) return;

    cout << root->value << " ";

    printPrefix(root->left);
    printPrefix(root->right);
}

// Функція для друку дерева у постфіксній формі (Reverse Polish notation)
void printPostfix(Node* root) {
    if (!root) return;

    printPostfix(root->left);
    printPostfix(root->right);

    cout << root->value << " ";
}

string readFormulaFromFile(const string& filename) {
    FILE* file = nullptr;
    fopen_s(&file, filename.c_str(), "r");  // Відкриваємо файл для читання (режим "r")

    if (!file) {  // Перевірка на помилку відкриття файлу
        cout << "Error opening file!" << endl;
        return "";
    }

    string formula;
    char ch;
```

```
    // Читаємо файл по символу
    while ((ch = fgetc(file)) != EOF) {
        formula += ch;
    }

    fclose(file);
    return formula;
}

int main() {
    string filename = "formula.txt";  // Назва файлу, що містить формулу
    string formula = readFormulaFromFile(filename);  // Читаємо формулу з файлу

    Node* root = new Node();

    int position = 0;
    buildTree(root, formula, position);

    // Друк дерева
    cout << "The tree is built: " << endl;
    printTree(root);

    // Обчислення дерева
    double result = evaluateTree(root);
    cout << "Result of the expression: " << result << endl;

    // Друк у префіксній формі
    cout << "Prefix notation: ";
    printPrefix(root);
    cout << endl;

    // Друк у постфіксній формі
    cout << "Postfix notation: ";
    printPostfix(root);
```

```
    cout << endl;


    return 0;
}
```

Вхідні данні:



Результат роботи коду:

```
The tree is built:
        3
    -
        8
/
        6
    +
        4
Result of the expression: 2
Prefix notation: / + 4 6 - 8 3
Postfix notation: 4 6 + 8 3 - /
```

Пояснення коду та операцій:

1. Структура Node:
   1.1. Призначення: Це вузол дерева. Він зберігає значення (число або оператор) та посилання на два піддерева — ліве та праве.
   1.2. Наприклад, для формули (4 + 6) / (8 - 3):
      1.2.1. Корінь дерева: /.
      1.2.2. Ліве піддерево: + з листами 4 та 6.
      1.2.3. Праве піддерево: - з листами 8 та 3.

2. Функція buildTree:
   2.1. Призначення: Рекурсивно будує дерево з математичного виразу.
   2.2. Вхідні параметри:
      2.2.1. Node*& root: корінь дерева (вказівник, який змінюється під час виконання).
      2.2.2. const string& formula: математичний вираз, поданий як рядок.

2.2.3. int& pos: позиція в рядку, з якої обробляється поточний символ.

2.3. Робота функції:

2.3.1. Пропускає пробіли.

2.3.2. Якщо зустрічає:

2.3.2.1. Число:

2.3.2.1.1. Зчитує послідовність цифр.

2.3.2.1.2. Створює новий вузол з цим числом.

2.3.2.1.3. Додає його до left або right в залежності від поточного стану дерева.

2.3.2.2. Оператор:

2.3.2.2.1. Зберігає оператор у поточний вузол дерева.

2.3.2.3. Відкриваюча дужка (:

2.3.2.3.1. Рекурсивно викликає buildTree для побудови піддерева.

2.3.2.4. Закриваюча дужка):

2.3.2.4.1. Завершує поточну рекурсію, повертаючи контроль функції.

2.3.3. Повторює обхід дерева через механізм restart.

3. Функція printTree:

3.1. Призначення: Друкує дерево в ієрархічному вигляді.

3.2. Робота:

3.2.1. Рекурсивно обходить дерево:

3.2.2. Спочатку праве піддерево (для візуалізації справа наліво).

3.2.3. Потім друкує поточний вузол (з урахуванням відступів, пропорційних глибині).

3.2.4. Нарешті, обробляє ліве піддерево.

3.3. Це дозволяє побачити структуру дерева "зліва на право".

4. Функція evaluateTree:

4.1. Призначення: Обчислює значення математичного виразу, поданого у вигляді дерева.

4.2. Робота:

4.2.1. Базовий випадок: якщо вузол є листом (немає дітей), повертає його числове значення.

4.2.2. Рекурсивно обчислює значення лівого та правого піддерева.

4.2.3. Виконує математичну операцію, вказану у вузлі:

4.2.3.1. +: додає.

4.2.3.2. -: віднімає.

4.2.3.3. *: множить.

4.2.3.4. /: ділить (із перевіркою на ділення на нуль).

4.2.4. У разі невідомого оператора або помилки повертає значення -999999999 і виводить повідомлення про помилку.

5. Функція printPrefix:

5.1. Призначення: Друкує дерево у префіксній формі (Polish notation).

5.2. Робота:

5.2.1. Друкує поточний вузол.

5.2.2. Рекурсивно обробляє ліве піддерево.

5.2.3. Рекурсивно обробляє праве піддерево.

6. Функція printPostfix:

6.1. Призначення: Друкує дерево у постфіксній формі (Reverse Polish notation).

6.2. Робота:

6.2.1. Рекурсивно обробляє ліве піддерево.

6.2.2. Рекурсивно обробляє праве піддерево.

6.2.3. Друкує поточний вузол.

7. Функція readFormulaFromFile:

7.1. Відкриває файл: fopen_s для відкриття файлу для читання.

7.2. Перевірка помилки: Якщо файл не відкрився, виведе повідомлення про помилку.

7.3. Читання вмісту: Читає файл символ за символом, додаючи їх до рядка.

7.4. Закриття файлу: Після завершення роботи закриває файл.

7.5. Повернення результату: Повертає рядок, що містить вміст файлу.

8. Функція main:

8.1. Читає вираз з файлу у string.

8.2. Будує дерево за допомогою buildTree.

8.3. Друкує дерево у структурному вигляді через printTree.

8.4. Обчислює результат виразу за допомогою evaluateTree.

8.5. Друкує дерево у префіксній та постфіксній формах.