Отчет по лабораторной работе № 24 по курсу

«Практикум программирования»

Студент группы М8О-109Б-22 Ефименко Кирилл Игоревич

Контакты: telegram @vivichv9

Работа выполнена: 20.05.2023

Преподаватель: каф.806 Сысоев Максим Алексеевич

Отчет сдан «25» июня 2023 г., итоговая оценка ____

Подпись преподавателя _____

1. Тема: Деревья выражений

2. Цель работы: Построить и обработать дерево выражений, а так же выполнить специальное действие

3. Задание (вариант № 49):

$$49.\ \text{Дан многочлен } f(x) = \sum_{i=0}^{n} a_i x^i.\ \text{Построить многочлен } g(x) = \sum_{i=0}^{n} a_{n-i} x^i.$$

4. Оборудование (студента):

Процессор AMD Ryzen 5 5600H with Radeon Graphics 3.30 GHz, ОП 16,0 Гб, SSD 512 Гб. Монитор 1920x1080 144 Hz

5. Программное обеспечение (студента):

Операционная система семейства Linux, наименование Ubuntu, версия 18.10

Интерпретатор команд: bash, версия 4.4.19

Система программирования – версия --, редактор текстов Emacs, версия 25.2.2

Утилиты операционной системы –

Прикладные системы и программы –

Местонахождение и имена файлов программ и данных на домашнем компьютере –

6. Идея, метод, алгоритм решения задачи *(в формах: словесной, псевдокода, графической [блок-схема, диаграмма, рисунок, таблица] или формальные спецификации с пред- и постусловиями)*

Строю дерево выражений, затем строю поддерево для выражения, на которое необходимо заменить, ищу переменную в дереве и произвожу замену

7. Сценарий выполнения работы *(план работы, первоначальный текст программы в черновике [можно на отдельном листе] и тесты, либо соображения по тестированию)*

1. Читаю про деревья выражений
2. Пишу алгоритм Дейкстры для польской записи
3. Делаю дерево и пишу код функции специального действия

8. Распечатка протокола *(подклеить листинг окончательного варианта программы с тестовыми примерами, подписанный преподавателем)*

Node.hpp

```hpp
#ifndef NODE_HPP_INCLUDED
#define NODE_HPP_INCLUDED

class Tree;

class Node {
    friend class Tree;

private:
    char data;
    Node* left;
    Node* right;

public:
    Node();
    Node(const char data);
    Node(const char data, Node* left, Node* right);
};

#include "../src/Node.cpp"

#endif
```

Node.cpp

```cpp
#include "../include/Node.hpp"

Node::Node(): data(0), left(nullptr), right(nullptr) {}

Node::Node(const char data): data(data), left(nullptr), right(nullptr) {}

Node::Node(const char data, Node* left, Node* right): data(data), left(left), right(right) {}
```

Stack.hpp

```hpp
#ifndef STACK_HPP_INCLUDED
#define STACK_HPP_INCLUDED

#include "../../KP7/include/Vector.hpp"
#include <iostream>

template <typename T>
class Stack {
private:
    Vector<T> stack;

public:
    Stack() = default;
    Stack(const std::initializer_list<T>& list);
    ~Stack() = default;

    size_t size() const;
    bool empty() const;
    void push(const T& value);
    void pop();
    T& top();
    const T& top() const;
};

#include "../src/Stack.cpp"

#endif
```

Stack.cpp

```cpp
#include "../include/Stack.hpp"

template <typename T>
Stack<T>::Stack(const std::initializer_list<T>& list) {
    for (T elem: list) {
        stack.push_back(elem);
    }
}

template <typename T>
size_t Stack<T>::size() const {
    return stack.get_size();
}

template <typename T>
bool Stack<T>::empty() const {
    return stack.get_size() == 0;
}

template <typename T>
void Stack<T>::push(const T& value) {
    stack.push_back(value);
}

template <typename T>
void Stack<T>::pop() {
    stack.pop_back();
}

template <typename T>
T& Stack<T>::top() {
    return stack.back();
}
```

```cpp

template <typename T>
const T& Stack<T>::top() const {
    return stack.back();
}
```

Tree.hpp

```cpp
#ifndef TREE_HPP_INCLUDED_
#define TREE_HPP_INCLUDED_

#include "Node.hpp"
#include "Stack.hpp"
#include <iostream>
#include <string>

class Tree {
private:
    Node* root;

public:
    Tree();
    Tree(const std::string& expression);
    ~Tree();

    Node* get_root() const;

    void delete_tree(Node* node);
    Node* create_tree(const std::string& postfix);
    std::string to_postfix(const std::string& expression);

    void print_tree(Node* root, const size_t height = 0) const;
    void task(Node* root);
};

#endif
```

Tree.cpp

```cpp
#include "../include/Tree.hpp"

Tree::Tree(): root(nullptr) {}

Tree::Tree(const std::string& expression) {
    std::string postfix = to_postfix(expression);
    root = create_tree(postfix);
}

Tree::~Tree() {
    delete_tree(root);
}

Node* Tree::get_root() const {
    return this->root;
}

bool isOperator(char c) {
    return (c == '+' || c == '-');
}

size_t getPriority(char c) {
    if (c == '+' || c == '-') {
        return 1;
    }

    return 0;
}
```

```cpp
std::string Tree::to_postfix(const std::string& expression) {
    std::string postfix = "";
    Stack<char> stack;

    for (size_t i = 0; i != expression.size(); ++i) {
        char c = expression[i];

        if (!isOperator(c) && c != '(' && c != ')') {
            postfix += c;
        } else if (c == '('){
            stack.push(c);
        } else if (c == ')'){
            while (stack.top() != '('){
                postfix += stack.top();
                stack.pop();
            }
            stack.pop();
        } else {
            while (!stack.empty() && (getPriority(stack.top()) >= getPriority(c))){
                postfix += stack.top();
                stack.pop();
            }
            stack.push(c);
        }
    }

    while (!stack.empty()) {
        postfix += stack.top();
        stack.pop();
    }
    return postfix;
}
```

```cpp
void Tree::delete_tree(Node* node) {
    if (node == nullptr) return;

    delete_tree(node->left);
    delete_tree(node->right);
    delete node;
}

```

```cpp
Node* Tree::create_tree(const std::string& postfix) {
    if (postfix.length() == 0) return nullptr;

    Stack<Node*> stack;

    for (char c: postfix) {
        if (isOperator(c)) {
            Node* node_x = stack.top();
            stack.pop();

            Node* node_y = stack.top();
            stack.pop();

            Node* node = new Node(c, node_y, node_x);
            stack.push(node);
        } else {
            stack.push(new Node(c));
        }
    }

    return stack.top();
}

void Tree::print_tree(Node* root, const size_t height) const {
    if (root != nullptr){
        print_tree(root->right, height + 1);
        for (size_t i = 0; i < height; ++i) {
            std::cout << "\t";
        }
        std::cout << root->data << "\n";
        print_tree(root->left, height + 1);
    }
}
```

```cpp
void Tree::task(Node* root) {
    if (root == nullptr) {
        return;
    }
    task(root->right);
    std::cout << root->data;
    task(root->left);
}
```

main.cpp

```cpp
#include "Tree.hpp"
#include "../src/Tree.cpp"

int main() {
    std::string expr = "2+3+1+4";
    Tree tree(expr);
    tree.task(tree.get_root());
}
```

9. Дневник отладки *(дата и время сеансов отладки и основные события [ошибки в сценарии и программе, нестандартные ситуации] и краткие комментарии к ним. В дневнике отладки приводятся сведения об использовании других ЭВМ, существенном участии преподавателя и других лиц в написании и отладке программы)*

№	*Лаб. или дом*	*Дата*	*Время*	*Событие*	*Действие по исправлению*	*Примечания*

Проблем при выполнении лабы не возникло

10. Замечания автора *(по существу работы)*

Замечания отсутствуют

11. Вывод

При выполнении лабораторной познакомился с деревьями выражений. В целом довольно интересная вещь, особенно реализация, но нет идей, где получится применить их на практике. Получилось самому реализовать алгоритм Дейкстры для перевода в обратную польскую запись.

Подпись студента _____