Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Кафедра инженерной психологии и эргономики

Основы алгоритмизации и программирования

Отчет по лабораторной работе №14

««Обратная польская запись»

Выполнил: Усов А.М.
Студент группы 310901
Преподаватель: Кабариха В. А.

Минск 2023

Цель: сформировать знания и умения по работе с подпрограммами, приобрести навыки написания программ с использованием бинарных деревьев.

Задание 15. По бесскобочной постфиксной записи арифметического выражения с операндами — строками построить дерево выражения и получить полноскобочное инфиксное выражение.

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

using namespace std;
template <typename T>
class Tree {
	class TreeNode {
		T data;
		TreeNode* left;
		TreeNode* right;
	public:
		TreeNode(T data) : data(data), left(nullptr), right(nullptr) {}
		void setLeft(TreeNode* left) {
			this->left = left;
		}
		void setRight(TreeNode* right) {
			this->right = right;
		}
		T getData() {
			return data;
		}
		TreeNode* getLeft() {
```

```
            return left;
        }
        TreeNode* getRight() {
            return right;
        }
    };

    TreeNode* root;
public:
    Tree() : root(nullptr) {}


    void Create(string expression) {
        int countOperators = 1;

        for (int i = 0; i < expression.size(); i++) {
            if (expression[i] == ' ') continue;
            string s = "";
            while (expression[i] != ' ' && i < expression.size()) {
                s += expression[i];
                i++;
            }
            TreeNode* node = new TreeNode(s);
            if (isOperator(node->getData())) {
                node->setRight(root);
                TreeNode* needNode = root;

                if (isOperator(root->getData())) {
                    for (int j = 0; j < countOperators; j++) {
```

```
                                needNode = needNode->getLeft();
                                if (isOperator(needNode->getData())) {
                                        countOperators++;
                                }

                        }
                }
                else {
                        needNode = root;
                }

                node->setLeft(needNode->getRight());
                needNode->setRight(nullptr);
        }
        else {
                node->setRight(root);
        }
        root = node;
    }
}

string getNewExp() {
    return createExp(root);
}

string createExp(TreeNode* node) {
    if (node == nullptr) return "";
    if (node->getLeft() == nullptr && node->getRight() == nullptr) {
        return node->getData();
```

```
        }
        string left = createExp(node->getLeft());
        string right = createExp(node->getRight());
        //cout << left << "\t " << right  << "\t " << node->getData() << endl;
        int leftOperatorLevel = 3;
        int rightOperatorLevel = 3;
        if (isOperator(node->getData())) {
                leftOperatorLevel = getPriority(node->getLeft()->getData());
                rightOperatorLevel = getPriority(node->getRight()->getData());
        }
        int currentOperatorLevel = getPriority(node->getData());
        if (leftOperatorLevel < currentOperatorLevel) {
                left = "(" + left + ")";
        }
        if (rightOperatorLevel < currentOperatorLevel) {
                right = "(" + right + ")";
        }
        return left + " " + node->getData() + " " + right;




}

bool isOperator(string c) {
        return c == "+" || c == "-" || c == "*" || c == "/";
}

void Clear(TreeNode* node) {
```

```
            if(node == nullptr) return;
            Clear(node->getLeft());
            Clear(node->getRight());
            delete node;
        }

        TreeNode* getRoot() {
            return root;
        }
        int getPriority(string c) {
            if (c == "+" || c == "-") return 1;
            if (c == "*" || c == "/") return 2;
            return 3;
        }
};



int main()
{
        Tree<string> tree;


        string expression = "3 1 +";
        string expression2 = "1 2 - 3 4 /5 6 / + +";
        string expression3 = "1 2 + 3 4 - *";
        string expression4 = "1 2 + 1 * 4 6 + 2 * +";

        cout << expression << endl;
```

```
    tree.Create(expression);
    cout << tree.getNewExp() << endl << endl;

    Tree<string> tree2;
    cout << expression2 << endl;
    tree2.Create(expression2);
    cout << tree2.getNewExp() << endl << endl;

    Tree<string> tree3;
    cout << expression3 << endl;
    tree3.Create(expression3);
    cout << tree3.getNewExp() << endl << endl;

    Tree<string> tree4;
    cout << expression4 << endl;
    tree4.Create(expression4);
    cout << tree4.getNewExp() << endl << endl;



    Tree<string> tree5;
    cout << "enter reverse polish notation: ";
    string expression5;
    getline(cin, expression5);
    tree5.Create(expression5);
    cout << tree5.getNewExp() << endl << endl;

    return 0;
}
```

Результат работы программы представлен на рисунке 1.

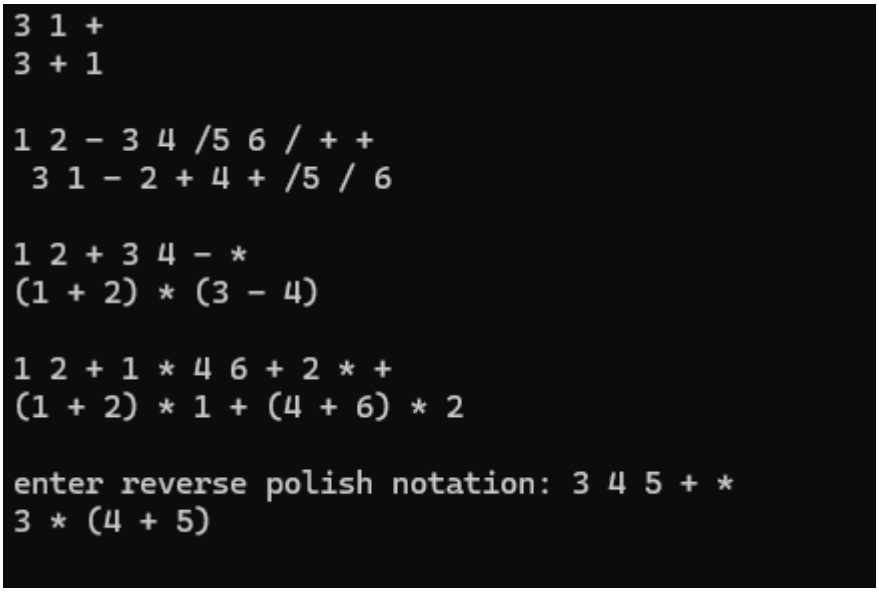

Рисунок 1 – Результат выполнения программы

Вывод: лабораторная работа №14 «Обратная польская запись» помогла мне сформировать знания и умения по работе с подпрограммами и бинарными деревьями, а также приобрести навыки написания программ с использованием бинарных деревьев. Полученные навыки будут полезны мне в дальнейшем при решении более сложных задач и разработке программных систем.