Отчет по лабораторной работе № 24 по курсу “Алгоритмы и структуры данных”

Студент группы М80-101Б-22, Шляхтуров Александр Викторович, № по списку 27

Контакты email: shliakhturov@gmail.com

Работа выполнена: «25» апреля 2023г.

Преподаватель: каф. 806 Крылов Сергей Сергеевич

Входной контроль знаний с оценкой _____

Отчет сдан « » _____202 __ г., итоговая оценка _____

Подпись преподавателя _____

1. **Тема:** Преобразование выражения в дерево
2. **Цель работы:** Научиться преобразовывать выражения в деревья и работать с ними

3. **Задание** Редуцировать выражение, заменив операцию умножение на сумму

4. **Оборудование**:

Оборудование ПЭВМ студента, если использовалось:

Процессор **AMD Ryzen 5 5500U 2.10 GHz, 6 ядер** с ОП **8192 Мб**, ТТН **512000 Мб**. Мониторы **Lenovo.**

5. **Программное обеспечение:**

Программное обеспечение ЭВМ студента, если использовалось:

Операционная система семейства **Linux**, наименование **Ubuntu** версия **20.04.5,** интерпретатор команд bash версия 5.0.17(1).

Система программирования CLion версия 2021.1.3

Редактор текстов nano версия 6.2

Утилиты операционной системы WinRar, Microsoft Word.

Прикладные системы и программы Ubuntu wsl, Clion, Google Chrome

Местонахождение и имена файлов программ и данных на домашнем компьютере /home/artur

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

Опишем структуру узла дерева, который будет хранить в себе в виде строки число или операцию: struct

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

Реализуем следующий функции:

int is_operator(char c) – будет возвращать 1, если символ – оператор

int is_num(string elem) – возвращает единицу, если строка это числа (возможно, отрицательное)

queue<string> get_revers_polish_notation(string input) – построение обратной польской нотации, по строке прочитанной из входного потока. Сначала эта строка преобразуется в очередь из чисел и операторов для удобного взаимодействия в дальнейшем. После при помощи алгоритма сортировочной станции Дейкстры, используя 2 очереди (входная, которую мы уже сформировали и выходную, которую по ходу будет формировать) и одного стека, в который будет помещать операторы.

Node* get_tree(queue<string> polish_notation) – «запихивание» обратной польской нотации в дерево выражения. Возвращает ссылку на root – корень дерева.

void print_tree(Node* node, int tab) – вывод дерева void get_infix(Node* node) – получение инфиксной формы при помощи ЛКП обхода.

int bin_pow(int x, int y) – бинарное возведение в степень

string calc_value(string el1, string el2, string oper) – выполнения действия, над двумя операндами, когда все они представленны в виде строк. void do_substract(Node* node, stack<string> & nums) – произведение вычитание из вершины, в которой расположен знак минус.
void search_substracts(Node* node) – обход дерева и поиск вершин с минусом.

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

Тесты:

a*7
a*3
a*1
a

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

```
#include<bits/stdc++.h>


using namespace std;


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


int is_operator(char c) {
    return c == '-' or c == '+' or c == '*' or c == '/' or c == '(' or c == ')' or c == '^';
}


int is_num(string elem) {
    return elem.size() >= 2 or isdigit(elem[0]) or isalpha(elem[0]);
}


queue<string> get_revers_polish_notation(string input) {
    queue<string> out;
    queue<string> in;
    stack<string> operators;
    string num;
    int in_num = 0;
    int was_operator = 0;
```

```
// creatin in queue
for (int i = 0; i < input.size(); i ++) {

    // if open skobka then was_op = 0
    if (input[i] == '(') {
        was_operator = 0;
    }

    // This is first digit of nums (positive)
    if (in_num == 0 and (isdigit(input[i]) or isalpha(input[i]))) {
        in_num = 1;
        num += input[i];
        continue;
    }

    // This is uno minus before num
    if (in_num == 0 and was_operator == 0 and input[i] == '-') {
        num += '-';
        in_num = 1;
        was_operator = 1;
        continue;
    }

    // This is num but not first elem
    if (in_num and (isdigit(input[i]) or isalpha(input[i]))) {
        num += input[i];
        continue;
    }
```

```
    // if operator push ego v in
    if (is_operator(input[i])) {
        if (num.size()) {
            in.push(num);
            in_num = 0;
            num = "";
        }
        in.push(string(1, input[i]));
        continue;
    }
}
if (num.size()) {
    in.push(num);
}
map<string, int> priority = {
    {"-", 1},
    {"+", 1},
    {"*", 2},
    {"/", 2},
    {"^", 3}
};
while (!in.empty()) {
    string elem = in.front();
    in.pop();
    if (is_num(elem)) {
        out.push(elem);
    } else {
        if (operators.empty() or elem == "(") {
            operators.push(elem);
            continue;
```

```
            }
            if (elem == ")") {
                while (operators.top() != "(") {
                    out.push(operators.top());
                    operators.pop();
                }
                operators.pop();
                continue;
            }
            while (operators.size() and priority[elem] <= priority[operators.top()]) {
                out.push(operators.top());
                operators.pop();
            }
            operators.push(elem);
        }
    }
    while (operators.size()) {
        out.push(operators.top());
        operators.pop();
    }

    return out;
}

Node* get_tree(queue<string> polish_notation) {
    stack<Node*> nodes;
    while (!polish_notation.empty()) {
        string elem = polish_notation.front();
        polish_notation.pop();
```

```
        if (is_num(elem)) {
            Node* new_node = new Node(elem);
            nodes.push(new_node);
        } else {
            Node* el_right = nodes.top();
            nodes.pop();
            Node* el_left = nodes.top();
            nodes.pop();
            Node* new_node = new Node(elem);
            new_node->left = el_left;
            new_node->right = el_right;
            nodes.push(new_node);
        }
    }
    return nodes.top();
}
void print_tree(Node* node, int tab) {
    if (node == nullptr) return;
    print_tree(node->right, tab + 1);
    for (int i = 0; i < tab; i ++) {
        cout << '\t';
    }
    cout << node->value << endl;
    print_tree(node->left, tab + 1);
}

void get_infix(Node* node) {
    if (node == nullptr) return;
    if (node->value.size() == 1 and is_operator((node->value)[0])) {
        cout << "(";
```

```
    }
    get_infix(node->left);
    cout << node->value;
    get_infix(node->right);
    if (node->left != nullptr and node->right != nullptr) {
        cout << ")";
    }
}

int bin_pow(int x, int y) {
    int z = 1;
    while (y) {
        if (y & 1) z *= x;
        y = y >> 1;
        x *= x;
    }
    return z;
}

string calc_value(string el1, string el2, string oper) {
    int
res;
    int is_negative = 0;
    if (oper == "+") {
        res = stoi(el1) + stoi(el2);
    }
    if (oper == "-") {
        res = stoi(el1) - stoi(el2);
    }
    if (oper == "*") {
```

```
        res = stoi(el1) * stoi(el2);
    }
    if (oper == "/") {
        res = stoi(el1) / stoi(el2);
    }
    if (oper == "^") {
        res = bin_pow(stoi(el1), stoi(el2));
    }

    string result = "";
    if (res < 0) {
        is_negative = 1;
        res *= -1;
    }
    while (res > 0) {
        string
digit;
        if (res % 10 == 0) digit = "0";
        if (res % 10 == 1) digit = "1";
        if (res % 10 == 2) digit = "2";
        if (res % 10 == 3) digit = "3";
        if (res % 10 == 4) digit = "4";
        if (res % 10 == 5) digit = "5";
        if (res % 10 == 6) digit = "6";
        if (res % 10 == 7) digit = "7";
        if (res % 10 == 8) digit = "8";
        if (res % 10 == 9) digit = "9";
        result += digit;
        res = floor(res / 10);
    }
```

```
    if (is_negative) result += "-";     reverse(result.begin(), result.end());
    if (result.empty()) {
        result = "0";
    }
    return result;
}




void do_substract(Node* node, stack<string> & nums) {
    if (node == nullptr) return;
    do_substract(node->left, nums);
    do_substract(node->right, nums);
    if (is_num(node->value)) {
        nums.push(node->value);
    } else {
        string el2 = nums.top();
        nums.pop();
        string el1 = nums.top();
        nums.pop();
        node->left = nullptr;
        node->right = nullptr;
        node->value = calc_value(el1, el2, node->value);
        nums.push(node->value);
    }
}

void search_substracts(Node* node) {
    if (node == nullptr) return;
```

```
    search_substracts(node->left);
    search_substracts(node->right);
    if (node->value == "-") {
        stack<string> nums;
        do_substract(node, nums);
    }
}


int main() {
    string input;
    string test;
    while (cin >> test) {
        input += test;
    }
    queue<string> polish = get_revers_polish_notation(input);
    Node* root = get_tree(polish);
    cout << endl << "Дерево выражения до преобразования" << endl << endl;
    print_tree(root, 0);
    cout << endl <<"Инфиксная запись до преобразования" << endl << endl;
    get_infix(root);
    queue<string> nums;
    search_substracts(root);
    cout << endl << endl  <<"Дерево выражения после преобразования" << endl << endl;
    print_tree(root, 0);
    cout << endl <<"Инфиксная запись после преобразования" << endl << endl;
    get_infix(root);
    cout << endl;
}
```

alexander@DESKTOP-KNBCFCI:~/lr24$ ./program < input1.txt
Дерево выражения до преобразования

```
      7
*
      a
```

Инфиксная запись до преобразования

(a*7)
Дерево выражения после преобразования

```
                              a
                         +
                              a
                    +
                         a
               +
                    a
          +
               a
     +
          a
+
     a
```

Инфиксная запись после преобразования

(a+(a+(a+(a+(a+(a+a))))))

```
                                   23
                -
                                   4
                          *
                                   22
```

Инфиксная запись до преобразования

(((((22*4)-(23*((3-8)^2)))+15)*5)-0)

Дерево выражения после преобразования

alexander@DESKTOP-KNBCFCI:~/lr24$ ./program < input1.txt
Дерево выражения до преобразования

a
Инфиксная запись до преобразования

a
Дерево выражения после преобразования

a
Инфиксная запись после преобразования

a

Инфиксная запись после преобразования

alexander@DESKTOP-KNBCFCI:~/lr24$ ./program < input1.txt
Дерево выражения до преобразования

    3
*
    a
Инфиксная запись до преобразования

(a*3)
Дерево выражения после преобразования

        a
    +
        a
+
    a
Инфиксная запись после преобразования

(a+(a+a))

9. **Дневник отладки** должен содержать дату 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.

10. **Замечания автора** по существу работы

11. **Выводы**

Я научился преобразовывать деревья в выражения и работать с ними. Узнал про алгоритм сортировочной станции Дейкстры и то, как происходит работа с математическими выражениями.

Недочёты при выполнении задания могут быть устранены следующим образом: --

Подпись студента

_____