МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра МО ЭВМ

ОТЧЕТ по

лабораторной работе №2

по дисциплине «Алгоритмы и структуры данных»

Тема: Рекурсивная обработка иерархических списков

Студент гр. 8304 Самакаев Д.И.

Преподаватель Фирсов М.А.

Санкт-Петербург

2019

Вариант 21

Цель работы.

Изучить основные принципы работы иерархических списков и их обработки.

Постановка задачи.

1. Изучив условия задачи разработать эффективный алгоритм обработки входных данных.

2. Сопоставить рекурсивное решение с итеративным решением задачи;

3. Сделать вывод о целесообразности и эффективности реализованного алгоритма решения задачи.

Пусть выражение (логическое, арифметическое, алгебраическое*) представлено иерархическим списком. В выражение входят переменные, которые являются атомами списка. Операции представляются в префиксной форме ((<операция> <аргументы>)), либо в постфиксной форме (<аргументы> <операция>)). Аргументов может быть 1, 2 и более. Например (в префиксной форме): (+ a (* b (- c))) или (OR a (AND b (NOT c))).

В задании даётся один из следующих вариантов требуемого действия с выражением: проверка синтаксической корректности, (преобразование), вычисление. Пример упрощения: (+ 0 (* 1 (+ a b))) преобразуется в (+ a b). В задаче вычисления на входе дополнительн задаётся список значений переменных ((x1 c1) (x2 c2) … (xk ck)), где xi – переменная, а ci - её значение (константа).

В индивидуальном задании указывается: тип выражения (возможно дополнительно – состав операций), вариант действия и форма записи.

* - здесь примем такую терминологию: в арифметическое выражение входят операции +, -, *, /, а в алгебраическое – +, -, * и дополнительно некоторые функции.

21) арифметическое, вычисление, постфиксная форма

Описание алгоритма.

На каждом шаге алгоритма к двум первым аргументам в выражении применяется операция, после чего эти аргументы заменяются на результат

применения к ним операции до тех пор, пока выражение не сведётся к простейшему виду.

Спецификация программы.

Программа предназначена для арифметического вычисления значения выражений, представленных в постфиксной форме.

Программа написана на языке C++. Входные данные подаются в виде строк текстового файла или консольным вводом.

Описание функций.

1. bool is_brackets_correct(std::string &expression)

   Определяет, правильно ли в строке expression расставлены скобки.

2. void fill_map(std::string& values_str, std::map<std::string, int>& arguements_values_map)

   Находит в строке values_str имена переменных и их значения, после чего заносит их в словарь arguments_values_map.

3. void reg_calc(std::string& expression)

   Основная функция, осуществляет вычисление результата выражения expression.

4. void substitute(std::string& expression, std::map<std::string, int>& arguements_values_map)

   Меняет в строке expression аргументы на их значения.

Вывод.

Была реализована программа, позволяющая вычислять значения арифметических выражений в постфиксной форме.

Приложение.

1)Тестирование.

```
1 : ((a b + ) ci + )
((a wow)(b 222)(ci 313))
Incorrect arguement value!
You didn't set all values
Final reuslt is ((a222+)313+)
2 : (((a + ) bi + ) c + )
((a 123)(bi 222)(c 313))
Final reuslt is 658
3 : ((a b -) (c d +) *)
((a 228)(b 150)(c 114)(d 114))
Final reuslt is 17784
4 : (ai b c + )
((ai 123)(b 222)(c 313))
Final reuslt is 658
5 : (a + )
((a 123))
Final reuslt is 123
6 : ((((a - ) b - ) c - ) d - )
((a 123)(b 222)(c 313)(d 424))
Final reuslt is -1082
7 : (a b c - )
((a 123)(b 222)(c 313))
Final reuslt is -412
8 : (a - )
((a 123))
Final reuslt is -123
9 : ((a b *) c *)
((a 123)(b 222)(c 313))
Final reuslt is 8546778
10 : (((a *)b *) c *)
((a 123)(b 222)(c 313))
Final reuslt is 8546778
11 : (a b c *)
((a 123)(b 222)(c 313))
Final reuslt is 8546778
12 : (a *)
((a 123))
Final reuslt is 123
13 : ((a b /) c /)
((a 123)(b 23)(c 3))
Final reuslt is 1
14 : (((a /) b /) c /)
((a 123)(b 23)(c 3))
Final reuslt is 1
15 : (a b c /)
((a 123)(b 23)(c 3))
Final reuslt is 1
16 : (a /)
((a 123))
Final reuslt is 123
```

((a b +) ci +) ((a wow)(b 222)(ci 313))	Incorrect argument value! You didn’t set all values ((a222+)313+)
(((a +) bi +) c +)((a 123)(bi 222)(c 313))	658
((a b -) (c d +) *) ((a 228)(b 150)(c 114)(d 114))	17784
(ai b c +) ((ai 123)(b 222)(c 313))	658
(a +) ((a 123))	123
((((a -) b -) c -) d -) ((a 123)(b 222)(c 313)(d 424))	-1082
(a b c -) ((a 123)(b 222)(c 313))	-412
(a -) ((a 123))	-123
((a b *) c *) ((a 123)(b 222)(c 313))	8536778
(((a *)b *) c *) ((a 123)(b 222)(c 313))	8536778
(a b c *) ((a 123)(b 222)(c 313))	8536778
(a *) ((a 123))	123
((a b /) c /) ((a 123)(b 23)(c 3))	48
(((a /) b /) c /) ((a 123)(b 23)(c 3))	943
(a b c /) ((a 123)(b 23)(c 3))	1
(a /) ((a 123))	123

2)Исходный код.

```
#include <iostream>
#include <regex>
#include <string>
#include <fstream>
#include <map>


bool is_brackets_correct(std::string &expression) {

	int brackets_cnt = 0;

	for (size_t i = 0; i < expression.length(); i++) {
		if (brackets_cnt < 0)
			return false;
		else {
			if (expression[i] == '(')
				brackets_cnt++;
			else if (expression[i] == ')')
```

```
                        brackets_cnt--;
                    else continue;
                }
        }
        if (brackets_cnt == 0)
                return true;
        else return false;
}

void fill_map(std::string& values_str, std::map<std::string, int>& arguements_values_map) {

        std::regex pattern("(\\((\\w+) (-?\\d+)\\))");
        std::smatch match;

        int is_like_pattern = 0;
        while (is_like_pattern = std::regex_search(values_str, match, pattern)) {
                arguements_values_map.insert(std::pair<std::string, int>(match[2].str(),
stoi(match[3])));
                values_str.erase(match.position(2), match[2].length());
        }

        //std::cout << "Arguements values" << std::endl;

        for (int i = 0; i < values_str.length(); i++) {
                if (isalpha(values_str[i])) {
                        std::cout << "Incorrect arguement value!" << std::endl;
                        break;
                }
        }

        /*for (auto it = arguements_values_map.begin(); it != arguements_values_map.end(); ++it)
{
                std::cout << it->first << " = " << it->second << std::endl;
        }*/
}

void reg_calc(std::string& expression) {
        //group №         0,1,2     3,4          5   6,7      8         9        10         11
        std::regex patt("\\(( ){0,}((\\-)?\\d+)( )+)(( ){0,}(\\-?\\d+)( )+){0,}([\\-+*\\/])(
){0,}\\)");

        std::smatch match;
        int is_like_pattern = 0;
        int calc_res = 0;
        bool is_binary = false;
        int space_index = -1;

        //std::cout << "Calculation:" << std::endl;
        while (is_like_pattern = std::regex_search(expression, match, patt)) {

                std::string str_to_insert;
                if (match[10] == '+') {
                        //std::cout << expression << std::endl;
                        if (!match[6].matched) {
                                str_to_insert = " " + match[3].str() + " ";
                                expression.replace(match.position(0), match[0].length(),
str_to_insert);
                        }
                        else {
                                calc_res = stoi(match[3]) + stoi(match[8]);
                                str_to_insert = " " + std::to_string(calc_res) + " ";
                                expression.erase(match.position(8), match[8].length());
                                expression.replace(match.position(1), match[1].length(),
str_to_insert);
                        }
```

```
            }
            else if (match[10] == '-') {
                  //std::cout << expression << std::endl;
                  if (!match[6].matched) {
                        str_to_insert = " " + match[3].str() + " ";
                        expression.replace(match.position(0), match[0].length(),
str_to_insert);
                        if (!is_binary) {
                              if (match[4].matched)
                                    expression.erase(match.position(4) - 1, 1);
                              else
                                    expression.insert(match.position(3), "-");
                        }
                        is_binary = true;
                  }
                  else {
                        calc_res = stoi(match[3]) - stoi(match[8]);
                        str_to_insert = " " + std::to_string(calc_res) + " ";
                        expression.erase(match.position(8), match[8].length());
                        expression.replace(match.position(1), match[1].length(),
str_to_insert);
                        is_binary = true;
                  }
            }
            else if (match[10] == '*') {
                  //std::cout << expression << std::endl;
                  if (!match[6].matched) {
                        str_to_insert = " " + match[3].str() + " ";
                        expression.replace(match.position(0), match[0].length(),
str_to_insert);
                  }
                  else {
                        calc_res = stoi(match[3]) * stoi(match[8]);
                        str_to_insert = " " + std::to_string(calc_res) + " ";
                        expression.erase(match.position(8), match[8].length());
                        expression.replace(match.position(1), match[1].length(),
str_to_insert);
                  }
            }
            else if (match[10] == '/') {
                  //std::cout << expression << std::endl;
                  if (match[8].str() == "0")
                        std::cout << "Devide by zero!" << std::endl;
                  else {
                        if (!match[6].matched) {
                              str_to_insert = " " + match[3].str() + " ";
                              expression.replace(match.position(0), match[0].length(),
str_to_insert);
                        }
                        else {
                              calc_res = stoi(match[3]) / stoi(match[8]);
                              str_to_insert = " " + std::to_string(calc_res) + " ";
                              expression.erase(match.position(8), match[8].length());
                              expression.replace(match.position(1), match[1].length(),
str_to_insert);
                        }
                  }
            }
      }

      space_index = expression.find(' ');
      while (space_index != -1) {
            expression.erase(space_index, 1);
            space_index = expression.find(' ');
      }
```

```
	std::cout << "Final reuslt is " << expression << std::endl;
}

//replace var names with their values
void substitute(std::string& expression, std::map<std::string, int>& arguements_values_map) {

	int pos = 0;
	std::string buff;
	for (auto it = arguements_values_map.begin(); it != arguements_values_map.end(); ++it) {
		buff = std::to_string(it->second);
		pos = expression.find(it->first);
		while (pos != -1) {
			if ((pos == 0 || !isalpha(expression[pos - 1])) && !isalpha(expression[pos
+ it->first.length()])) {
				expression.replace(pos, it->first.length(), buff);
				pos = expression.find(it->first, pos);
			}
			else pos = expression.find(it->first, pos + 1);
		}
	}

	arguements_values_map.clear();

	//std::cout << "replaced variables with their values:" << std::endl << expression <<
std::endl;

	for (int i = 0; i < expression.length(); i++) {
		if (isalpha(expression[i])) {
			std::cout << "You didn't set all values" << std::endl;
			return;
		}
	}

}

void console_input(std::map <std::string, int>& arguements_values_map) {

	std::cout << "Please enter an expression" << std::endl;
	std::string expression;

	getline(std::cin, expression);

	std::cout << "Please enter the values" << std::endl;
	std::string values;

	getline(std::cin, values);

	fill_map(values, arguements_values_map);

	if (is_brackets_correct(expression)) {
		substitute(expression, arguements_values_map);
		reg_calc(expression);
	}

	std::cout << expression << std::endl;

}

void file_input(char* argv, std::map <std::string, int>& arguements_values_map) {

	std::ifstream file;
	std::string file_name = argv;

	file.open(file_name);
```

```
	if (!file.is_open())
		std::cout << "Error! File isn't open" << std::endl;

	std::string expression;
	std::string values;
	int i = 0;
	while (!file.eof()) {
		i++;
		getline(file, expression);
		getline(file, values);

		std::cout << i << " : " << expression << std::endl << values << std::endl;

		fill_map(values, arguements_values_map);

		if (is_brackets_correct(expression)) {
			//std::cout << expression << std::endl;
			substitute(expression, arguements_values_map);
			reg_calc(expression);
			//std::cout << std::endl;
		}
	}
}

int main(int argc, char** argv)
{
	std::map <std::string, int> arguements_values_map;
	if (argc == 1)
		console_input(arguements_values_map);
	else if (argc == 2)
		file_input(argv[1], arguements_values_map);
	else std::cout << "Please check arguments are correct";
}
```