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: Иерархические списки

Студент гр. 9304 _____ Борисовский В. Ю.

Преподаватель _____ Филатов А. Ю.

Санкт-Петербург

2020

Цель работы.

Изучить иерархические списки, написать программу, которая взаимодействует с ними.

Задание.

Вариант 13

Вычислить глубину (число уровней вложения) иерархического списка как максимальное число одновременно открытых левых скобок в сокращённой скобочной записи списка; принять, что глубина пустого списка и глубина атомарного S-выражения равны нулю; например, глубина списка (a (b ( ) c) d) равна двум;) равна двум;

Выполнение работы.

1) Сперва я создал структуру Node, которая предназначена для хранения одного узла иерархического списка. Полями данной структуры являются Node *next - указатель на следующий элемент списка и поле std::variant<Node*, std::string> val - данное поле может быть либо типа string либо типа Node*.

2) После этого была написана функция bool create_list(Node **head, std::string &str, int &index) создающая из строки иерархический список. Данная функция является рукурсивной. Принцип ее работы заключается в том, что она последовательно идет по строке (есть проверка на то, что строка обязательно должна начинаться с «(», иначе функция вернет false, что сигнализирует о некорректности данных) и встречая элемент, то есть букву, создается узел Node, если это первый созданный элемент то им инциализируется голова списка head, иначе же в цикле проходимся до последнего созданного на этом уровне элемента и присваиваем его полю next только что созданный узел. Поле val иницализируется буквой, которую соответственно встретили. После чего увеличиваем index и идем дальше. В

случае если встречаем пробел просто увеличиваем index. Если же мы встречаем открывающую скобку, то рекурсивно вызываем функцию create_list, а ее результат присваивем в булевую переменную checker. Затем выполняем проверку на то корректно ли отработа функция, если нет, то возвращаем false. Таким образом проходимся по всей строке, возвращая false в ситуациях которые не могут быть в строке (невалдиная строка).

3) Затем была написана функция int levels_counter(Node* head, int&counter), которая подсчитывает уровень вложенности списка. Работает она просто. В функции реализован обохд по дереву и если на одном уровне мы встретим хотя бы одну букву и поле next этого узла будет HE NULL, то можем утверждать, что этот уровень не является пустым списком и не является атомарным S - выражением (то есть, не состоит из одного элемента), после чего инкрементируем счетчик counter, и устанваливаем флаг в положение 0, для того чтобы не инкрементировать переменную на одном уровне дважды.

4) Для очистки памяти была реализована void deleter(Node *head). Она рекурсивно проходится по каждому узлу списка и вызывает для него deleter.

5) В заключение была релазиована функция main. В началае в булевую переменную записывается результат работы функции create_list, затем если функция отработала корректно будет вызвана функция levels_counter и выведен уровень вложенности списка. Иначе же будет выведено сообщение, о неккоректности введеных данных. В конце концов очищается вся выделенная память функцией deleter, если она вообще выделялась. Ввод аргументов реализован черз cli - command line interface.

Тестирование.

Запуск программы начинается с ввода команды “make”, что приведёт к компиляции программы и созданию исполняемого файла lab2. Запуск

программы производится командой ./lab2” и последующим вводом строки, содержащей логическое выражение. Тестирование производится с помощью скрипта test_skript.py. Запуск скрипта производится командой «python3 test_skript.py» в директории lab2.

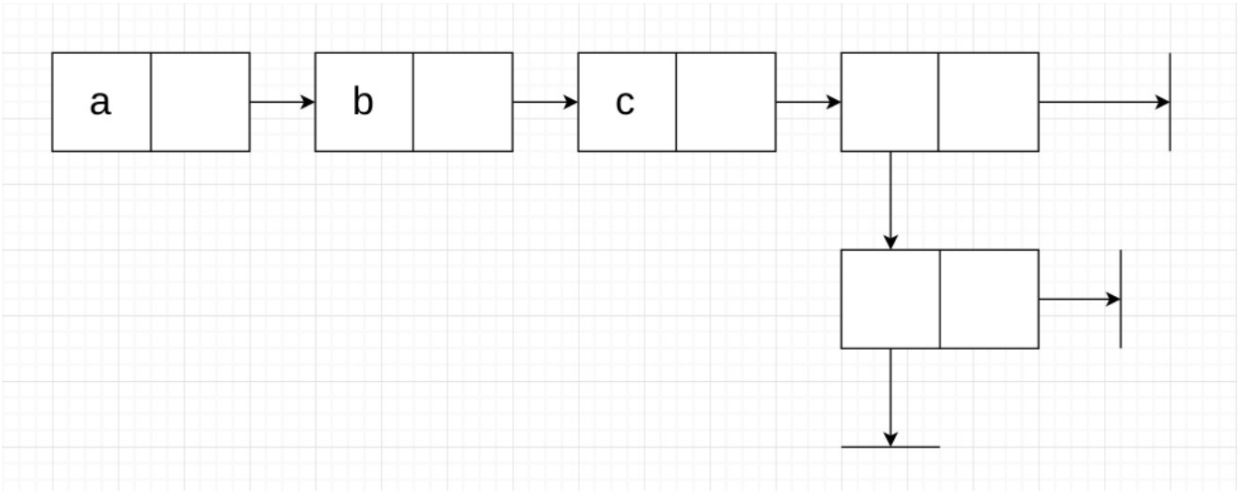
Результаты тестирования представлены в приложении Б.

Выводы.

Ознакомились с иерархическими списками и особенностями их программирования на языке программирования C++, используя знания о рекурсии. Реализовали программу, которая считывает входную строку в виде иерархического списка и подсчитывает его глубину вложенности.

Визуализация списка.

Для примера возьмем список (a b c ()).



ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <variant>


struct Node{
    Node* next;
    std::variant<Node*, std::string> val;
};

    void deleter(Node *head);


    bool create_list(Node **head, std::string &str, int &index){
        bool check = true;
        if (str[index] != '('){
            return false;
        }

        index++;

        while(str[index] != ')' && index < str.length()){
            if (isalpha(str[index]) && (str[index - 1] == '(' || str[index - 1] == ' ')){
                std::string elem = "";
                elem += str[index];
                if (!*head){
                    *head = new Node;
                    Node *ptr = *head;
                    ptr -> val = elem;
                    ptr -> next = nullptr;
                    index++;
                } else {
                    Node *ptr = *head;
                    while (ptr -> next){
                        ptr = ptr -> next;
                    }
                    ptr -> next = new Node;
                    ptr -> next -> val = elem;
                    ptr -> next -> next = nullptr;
                    index++;
                }
            }

             else if (str[index] == ' ' && (isalpha(str[index - 1]) || str[index - 1] == ')')){
                index++;
            }

```

```
            else if(str[index] == '(' && (str[index - 1] == ' '
  || str[index - 1] == '('){
                if (!*head){
                    *head = new Node;
                    Node *ptr = *head;
                    ptr -> val = nullptr;
                    ptr -> next = nullptr;
                     check = create_list(&std::get<Node*>(ptr ->
  val), str, index);
                    if (!check){
                        return false;
                    }
                    index++;
                } else {
                    Node *ptr = *head;
                    while (ptr -> next){
                        ptr = ptr -> next;
                    }
                    ptr -> next = new Node;
                    ptr -> next -> val = nullptr;
                    ptr -> next -> next = nullptr;
                     check = create_list(&std::get<Node*>(ptr ->
  next -> val), str, index);
                    if (!check){
                        return false;
                    }
                    index++;
                }
            }

            else{
                index = str.length();
                return false;
            }
        }
        if (index == str.length()){
            return false;
        } else {
            return true;
        }

    }

    int levels_counter(Node* head, int &counter){
        Node* ptr = head;
        int flag = 1;
        while (ptr){
             if (std::holds_alternative<std::string>(ptr -> val)
  && ptr ->next && flag){
                counter++;
                flag = 0;
            }
            if (std::holds_alternative<Node*>(ptr -> val)){
                if(std::get<Node*>(ptr -> val)){
                     levels_counter(std::get<Node*>(ptr -> val),
  counter);
```

```
                }
            }
            ptr = ptr -> next;
        }
        return counter;
    }

    void deleter(Node *head){
        if (head -> next){
            deleter (head -> next);
        }
        if (std::holds_alternative<Node*>(head -> val)){
            if (std::get<Node*>(head -> val)){
                deleter(std::get<Node*>(head -> val));
            }
        }
        delete head;
    }


    int main(int argc, char* argv[]) {
        if(argc == 1){
            std::cout << "Wrong expression\n";
            return 0;
        }

        int index = 0;
        int counter = 0;
        std::string str(argv[1]);
        Node *head = nullptr;
        bool checker = true;
        checker = create_list(&head, str, index);
        if (checker){
                        std::cout  <<  "This  list  have:  "  <<
  levels_counter(head, counter) << " level\n";
        } else {
            std::cout << "Wrong expression\n";
        }

        if(head){
            deleter(head);
        }

        return 0;
    }
```

ПРИЛОЖЕНИЕ Б

ТЕСТИРОВАНИЕ

Таблица Б.1 - Примеры тестовых случаев

№ п/п	Входные данные	Выходные данные	Комментарии
1.	(a b c ())	This list have: 1 level	Отработано успешно.
2.	(a s z ((())) (b a))	This list have: 2 level	Отработано успешно.
3.	()	This list have: 0 level	Отработано успешно.
4.	(() () ())	This list have: 0 level	Отработано успешно.
5.	(a b)	This list have: 1 level	Отработано успешно.
6.	(ab)	Wrong expression	Ошибка: не хватает пробела.
7.	( )	Wrong expression	Ошибка: лишний пробел.
8.	(a c v d ()	Wrong expression	Ошибка: не хватает закрывающей скобки.
9.		Wrong expression	Ошибка: выражение должно начинаться с «(»