

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

Фиалковский М.С.

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;

Выполнение работы.

Выполнение работы я разбил на несколько частей.

- 1) Сперва я создал структур *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 рекурсивной. 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 он инициализируется 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* инициализируется 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 присваиваем в булеву переменную *checker*. Затем выполняем проверку на то корректно ли отработала функция, если нет, то возвращаем *false*. Таким образом проходимся по всей строке, возвращая *false* в ситуациях которые не могут быть в строке (невалидная строка).

3) Затем была написана функция *int levels_counter(Node* head, int &counter)*, которая подсчитывает уровень вложенности списка. Работает она просто. В функции реализован обход по дереву и выполняется проверка на то, содержит ли *val* указатель на уровень ниже, если да то инкрементируется *counter*. После обхода всего списка возвращается значение переменной *counter*. Для того, чтобы скобки открытые на одном уровне не суммировались добавлена переменная *flag*, сигнализирующая о том, что на этом уровне *counter* уже инкрементировалась.

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

5) В заключение была реализована функция *main*. В начале в булеву 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 некорректности введенных данных. В конце концов очищается вся выделенная память функцией *deleter*, если она вообще выделялась. Ввод аргументов реализован через *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.

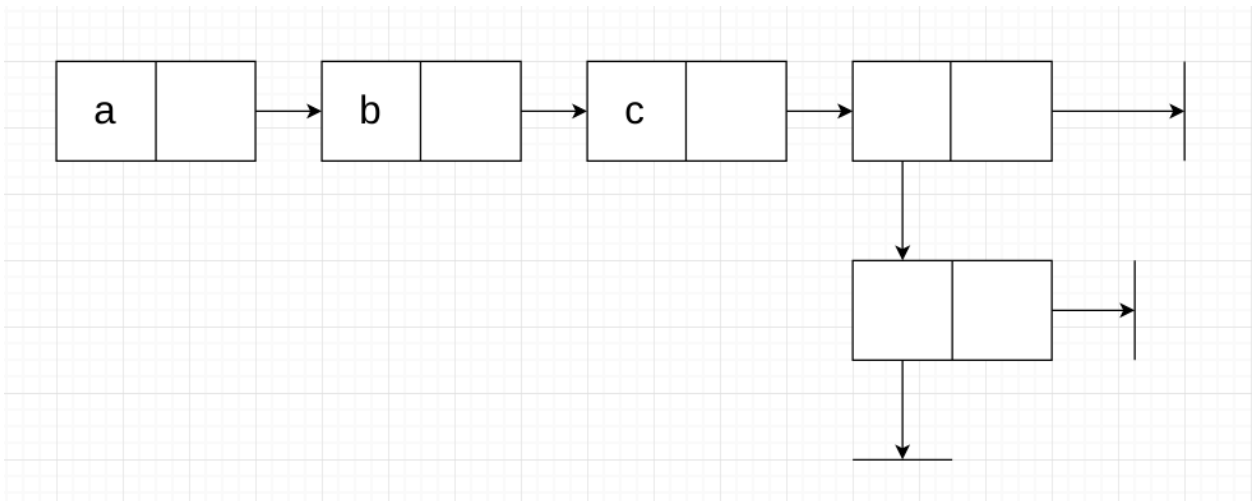
Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ 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: 3 level	Отработано успешно.
3.	()	This list have: 0 level	Отработано успешно.
4.	(() ())	This list have: 1 level	Отработано успешно.
5.	(a b)	This list have: 0 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 «(»

Визуализация списка.

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



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.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
1. #include <iostream>
2. #include <variant>
3.
4.
5.
6. struct Node{
7.     Node* next;
8.     std::variant<Node*, std::string> val;
9. };
10.
11. void deleter(Node *head);
12.
13.
14.
15. bool create_list(Node **head, std::string &str, int &index){
16.     bool check = true;
17.     if (str[index] != '('){
18.         return false;
19.     }
20.
21.     index++;
22.
23.     while(str[index] != ')' && index < str.length()){
24.         if (isalpha(str[index]) && (str[index - 1] == '(' || str[index - 1] == ' ')){
25.             std::string elem = "";
26.             elem += str[index];
```

```

27.     if (!*head){
28.         *head = new Node;
29.         Node *ptr = *head;
30.         ptr -> val = elem;
31.         ptr -> next = nullptr;
32.         index++;
33.     } else {
34.         Node *ptr = *head;
35.         while (ptr -> next){
36.             ptr = ptr -> next;
37.         }
38.         ptr -> next = new Node;
39.         ptr -> next -> val = elem;
40.         ptr -> next -> next = nullptr;
41.         index++;
42.     }
43. }
44.
45.     else if (str[index] == ' ' && (isalpha(str[index - 1]) || str[index - 1] ==
        '')){
46.         index++;
47.     }
48.
49.     else if(str[index] == '(' && (str[index - 1] == ' ' || str[index - 1] == '(')){
50.         if (!*head){
51.             *head = new Node;
52.             Node *ptr = *head;
53.             ptr -> val = nullptr;
54.             ptr -> next = nullptr;

```

```

55.         check = create_list(&std::get<Node*>(ptr -> val), str, index);
56.         if (!check){
57.             return false;
58.         }
59.         index++;
60.     } else {
61.         Node *ptr = *head;
62.         while (ptr -> next){
63.             ptr = ptr -> next;
64.         }
65.         ptr -> next = new Node;
66.         ptr -> next -> val = nullptr;
67.         ptr -> next -> next = nullptr;
68.         check = create_list(&std::get<Node*>(ptr -> next -> val), str,
index);
69.         if (!check){
70.             return false;
71.         }
72.         index++;
73.     }
74. }
75.
76. else{
77.     index = str.length();
78.     return false;
79. }
80. }
81. if (index == str.length()){
82.     return false;

```



```

83. } else {
84.     return true;
85. }
86.
87.}
88.
89.int levels_counter(Node* head, int &counter){
90.    Node* ptr = head;
91.    int flag = 1;
92.    while (ptr){
93.        if (std::holds_alternative<Node*>(ptr -> val)){
94.            if (flag){
95.                counter++;
96.            }
97.            flag = 0;
98.            if(std::get<Node*>(ptr -> val)){
99.                levels_counter(std::get<Node*>(ptr -> val), counter);
100.            }
101.        }
102.        ptr = ptr -> next;
103.    }
104.    return counter;
105. }
106.
107. void deleter(Node *head){
108.     if (head -> next){
109.         deleter (head -> next);
110.     }
111.     if (std::holds_alternative<Node*>(head -> val)){

```

```

112.         if (std::get<Node*>(head -> val)){
113.             deleter(std::get<Node*>(head -> val));
114.         }
115.     }
116.     delete head;
117. }
118.
119.
120. int main(int argc, char* argv[]) {
121.     if(argc == 1){
122.         std::cout << "Wrong expression\n";
123.         return 0;
124.     }
125.
126.     int index = 0;
127.     int counter = 0;
128.     std::string str(argv[1]);
129.     Node *head = nullptr;
130.     bool checker = true;
131.     checker = create_list(&head, str, index);
132.     if (checker){
133.         std::cout << "This list have: " << levels_counter(head, counter)
134.         << " level\n";
135.     } else {
136.         std::cout << "Wrong expression\n";
137.     }
138.
139.     if(head){
140.         deleter(head);

```

```
140.     }  
141.  
142.     return 0;  
143. }
```