

**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**

Студентка гр. 8381

Муковский Д.В.

Преподаватель

Жангиров Т.Р.

Санкт-Петербург

2019

### **Цель работы.**

Практическое применение и освоение иерархических списков, структур данных, функций для представления информации в памяти на языке программирования C++.

### **Задание (вариант №13).**

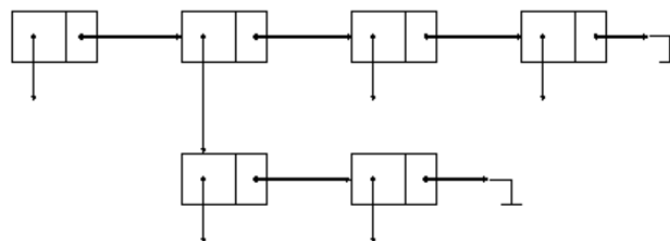
Вычислить глубину (число уровней вложения) иерархического списка как максимальное число одновременно открытых левых скобок в сокращённой скобочной записи списка; принять, что глубина пустого списка и глубина атомарного S-выражения равны нулю; например, глубина списка (a(b()c)d) равна двум.

### **Основные теоретические положения.**

Определим соответствующий тип данных  $S\_expr (El)$  рекурсивно, используя определение линейного списка (типа  $L\_list$ ):

$$\langle S\_expr (El) \rangle ::= \langle Atomic (El) \rangle \mid \langle L\_list (S\_expr (El)) \rangle,$$
$$\langle Atomic (E) \rangle ::= \langle El \rangle.$$
$$\langle L\_list(El) \rangle ::= \langle Null\_list \rangle \mid \langle Non\_null\_list(El) \rangle$$
$$\langle Null\_list \rangle ::= Nil$$
$$\langle Non\_null\_list(El) \rangle ::= \langle Pair(El) \rangle$$
$$\langle Pair(El) \rangle ::= ( \langle Head\_l(El) \rangle . \langle Tail\_l(El) \rangle )$$
$$\langle Head\_l(El) \rangle ::= \langle El \rangle$$
$$\langle Tail\_l(El) \rangle ::= \langle L\_list(El) \rangle$$

Традиционно иерархические списки представляют или графически или в виде скобочной записи. На рисунке приведен пример графического изображения иерархического списка. Соответствующая этому изображению сокращенная скобочная запись — это (a (b c) d e).



Переход от полной скобочной записи, соответствующей определению иерархического списка, к сокращенной производится путем отбрасывания конструкции Nil и удаления необходимое число раз пары скобок вместе с предшествующей открывающей скобке точкой.

Полная запись	Сокращенная запись
<i>a</i>	<i>a</i>
<i>Nil</i>	( )
<i>(a . (b . (c . Nil)))</i>	<i>(a b c)</i>
<i>(a . ((b . (c . Nil)) . (d . (e . Nil))))</i>	<i>(a (b c) d e)</i>

Согласно приведенному определению иерархического списка, структура непустого иерархического списка — это элемент размеченного объединения множества атомов и множества пар «голова-хвост».

### Выполнение работы.

Написание производилось на базе операционной системы Windows 10 в среде разработки Visual Studio.

Для реализации иерархического списка была реализована структура *S\_expr*(см. Приложение А) которая состоит из указателя на следующий элемент, флага(указывает что хранится в объединении) и объединение, которое содержит либо атом, либо указатель на элемент-наследник.

Считывание информации было реализовано тремя способами:

- 1) С помощью аргументов главной функции *main(argv u argc)*
- 2) Считывание из консоли
- 3) Считывание из файла input.txt

Далее строка подавалась в функцию *make\_list()*(см. Приложение А). Данная функция рекурсивно создавала иерархический список на основе строки. В случае открывающей скобки создавался элемент иерархического списка, который хранил в себе указатель на наследника, а в случае каких-либо символов создавался элемент иерархического списка, который хранил этот символ.

После создание иерархического списка, указатель на его голову подавался в функцию *deep\_rec()*, которая с помощью счетчика *count* считала текущую глубину списка. Функция была реализована следующим образом:

- 1) В случае если текущий элемент родитель, то счетчик увеличивался на один и функция вызывалась повторно, но с наследником.
- 2) В случае если текущий элемент родитель, то функция *deep\_rec* рекурсивно запускалась для следующего элемента, если он существует.
- 3) В случае если текущий элемент атом, то функция *deep\_rec* рекурсивно запускалась для следующего элемента, если он существует.
- 4) С помощью аргумента *answer*, которому присваивалось значение *count*, в случае если *count* > *answer*, то есть находилась наибольшая глубина.

Вывод результата производился с помощью функции *prints\_result()*, которая спрашивает у пользователя, хочет ли он записать результат в файл *output.txt*.

Таблица 1 – Тестирование программы

Входные данные	Результат
(a(b(dddd(fd()fdd)n)())()fdd))	The depth of the hierarchical list: 4
(o(m(e(g(a(lul))))())())	The depth of the hierarchical list: 5
(postav'()()()()((( )))()()()() zacet)	The depth of the hierarchical list: 1
(a)	The depth of the hierarchical list: 1

### Оценка сложности алгоритма.

Данный алгоритм имеет линейную зависимость от длины строки, то есть его сложность по времени оценивается как  $O(n)$ . Это обусловлено тем, что

функция *deep\_rec()* запускается единожды для каждого элемента иерархического списка.

### **Выводы.**

В процессе выполнения работы были получены практические навыки по применению структур иерархических списков, структур данных на языке программирования C++.

## Приложение А. Исходный код.

```
#include <iostream>
#include <cstdio>
#include <string>
#include <cctype>
#include <stack>
#include "conio.h"
#include <fstream>

using namespace std;

struct s_expr {
    int flag;
    s_expr* ptr_next;
    union {
        //реализация элемента иерархического списка
        char atom;
        s_expr* ptr_child;
    };
};

int is_right(string& arr) {
    stack<char> st;
    for (int i = 1; i < arr.length()-1; i++) {
        if (arr[i] == '(') {
            st.push(arr[i]);
            continue;
        }
        if (arr[i] == ')') {
            if (st.empty())
                //проверка правильности скобок
                return 1;
            char top_elem = st.top();
            if (top_elem == '(')
                st.pop();
            else return 1;
        }
    }
    if (st.empty())
        return 0;
    return 1;
}

s_expr* make_list(string& arr, int& count) {
    s_expr* head = nullptr;
    s_expr* last = nullptr;
    s_expr* cur = nullptr;
    while (arr[count]) {
        if (arr[count] == '(') {
            cur = new s_expr;
            if (!head) {
                head = cur;
                last = cur;
            }
        }
        else {
            last->ptr_next = cur;
            last = cur;
        }
        cur->flag = 0;
    }
}
```

```

        cur->ptr_next = nullptr;
        count++;
        cur->ptr_child = make_list(arr, count);
    }
    else
        if (arr[count] != '(' && arr[count] != ')') {
            if (!head) {
                head = new s_expr;
                head->flag = 1;
                head->ptr_next = nullptr;
                head->atom = arr[count];
                last = head;
                count++;
            }
            else {
                cur = new s_expr;
                last->ptr_next = cur;
                cur->flag = 1;
                cur->ptr_next = nullptr;
                cur->atom = arr[count];
                last = cur;
                count++;
            }
        }
    else
        if (arr[count] == ')') {
            count++;
            return head;
        }
    }
    return head;
}

void deep_rec(s_expr* cur, int& answer, int &count) {
    if (cur) {
        if (cur->flag == 0 && cur->ptr_child){
            count++;
            deep_rec(cur->ptr_child, answer, count);
            //функция нахождения глубины иерархического списка
        }
        if (count > answer) answer = count;
        if (cur->flag == 0 && !(cur->ptr_next) && !(cur->ptr_child)) count = 0;
        if (cur->flag == 1 && !(cur->ptr_next)) count = 0;
        deep_rec(cur->ptr_next, answer, count);
    }
}

int reads_data(string& arr) {
    cout << ">>Menu:\n>>Type '1' if you want to read data from the console"
        << "\n>>Type '2' if you want to read data from the file input.txt"
        << "\n>>Type 'Esc' if you want to exit" << endl;
    char key = _getch();
    if (key == '1') {
        cout << ">>Enter" << endl;
        getline(cin, arr);
        //Считывание из консоли
        if (!arr[0]) {
            cout << ">>\nWrong input";
            return 1;
        }
    }
}

```

```

    }
    else if (key == '2') {
        ifstream fs("input.txt", ios::in | ios::binary);
        if (!fs) {
            cout << ">>Error reading file\n";
            //Считывание из файла
            return 1;
        }
        getline(fs, arr);
        fs.close();
    }
    else if (key == 27){
        cout << ">>Goodbye\n";
        return 1;
    }
    else {
        cout<<">>Incorrect command, try again\n";
        reads_data(arr);
    }
    while (arr.find(' ') != string::npos)           //Удаление лишних пробелов
        arr.erase(arr.find(' '), 1);

    if (arr[0] != '(' && arr[arr.length() - 1] != ')') {
        cout << ">>\nWrong input";
        return 1;
    }

    //Проверка на корректность введенного списка
    if (is_right(arr)) {
        cout << ">>\nWrong input";
        return 1;
    }
    cout << "\n>>List:" << arr;
    return 0;
}

void prints_result(int answer, string &arr) {
    cout << "\n>>Type 'f' if you want to save result in file output.txt"
        << "\n>>Else type any button" << endl;
    char key = _getch();
    if (key == 'f') {
        std::ofstream out;
        out.open("output.txt");
        if (out.is_open()) {
            //Вывод результата в файл
            cout << "\n>>The depth of the hierarchical list: " << answer<<endl;
            out << ">>List: " << arr << '\n';
            out << ">>The depth of the hierarchical list: " << answer << endl;
            out.close();
            return;
        }
    }
    else {
        cout << "\n>>The depth of the hierarchical list: " << answer << endl;
        return;
    }
}

int main(int argc, char* argv[])
{

```



```

string arr;
if (argc > 1) {
    std::string arr_c(argv[1]);
    arr = arr_c;
}
else if (reads_data(arr)) return 0;
int count = 0;
s_expr* head = make_list(arr, count);
int answer = 0;
count = 0;
deep_rec(head, answer, count);
prints_result(answer, arr);
return 0;
}

```