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
Вариант 6

Студент гр. 8383 _____ Шишкин И.В.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2019

Цель работы.

Ознакомиться с основными понятиями и приёмами иерархических списков, улучшить навыки программирования рекурсивных функций на языке программирования С.

Постановка задачи.

6) проверить иерархический список на наличие в нем заданного элемента (атома) *x*;

Описание алгоритма.

Строка должна всегда начинаться открывающей скобкой и заканчиваться закрывающей. После считывания строки вызывается `func`, в которой строится по строке иерархический список. Затем вызывается функция `printAndCheck`, которая печатает атомы списка, а так же ищет в нем нужный атом, который ввел пользователь.

Описание структуры Node.

Структура Node содержит 4 поля: `char c` - символ, `struct Node *next` - указатель на следующий элемент в списке, `struct Node *child` - указатель на дочерний элемент в списке, `int isAtom` - переменная хранит 0, если элемент списка - скобка, и 1, если элемент списка атом.

Описание функции func.

Функция `Node *func(char *s, int count, int *flag)` получает на вход: `char *s` - строку, которую ввел пользователь, `int count` - счетчик глубины рекурсии, `int *flag` - счетчик открывающих и закрывающих скобок. Так же с помощью объявленной глобально переменной `int curr` происходит выбор нужного элемента из строки `s`. Если текущий элемент - открывающая скобка, то поле `isAtom` получает значение 0, значение `*flag` увеличивается на 1, и функция вызывается рекурсивно для дочернего списка. После вызова дочернего списка функция вызывается рекурсивно для

следующего элемента. Если же текущий элемент был не открывающей скобкой, а буквой, то в поле с структуры `Node` заносится текущий элемент, в `isAtom` устанавливается значение 1 и функция вызывается рекурсивно для следующего элемента. Если же текущий элемент был закрывающей скобкой, то значение `*flag` уменьшается на 1.

Описание функции printAndCheck.

Функция `void printAndCheckTree(Node *tree, char elem, int *flag)` получает на вход: дерево `Node *tree`, элемент, который нужно найти `char elem`, и `int *flag`, который хранит значение 1, если элемент удалось найти. С помощью цикла while функция проверяет все следующие элементы списка. Если у элемента есть дочерний список, то функция вызывается для него рекурсивно. Если в элементе списка поле `isAtom` хранит 1, то выводится элемент, хранящийся в поле `c`, а затем сравнивается с запрашиваемым элементом `elem`.

Спецификация программы.

Программа предназначена для поиска в иерархическом списке заданного атома. Программа написана на языке С. Входными данными является строка - считывается из файла или терминала. Итог работы программы выводится на экран. Атомами могут быть только английские буквы. Строка всегда начинается с открывающей скобки.

Иллюстрация построения списка по скобочной записи иерархического списка для случая (a(bc)de) см. на рис. 1.

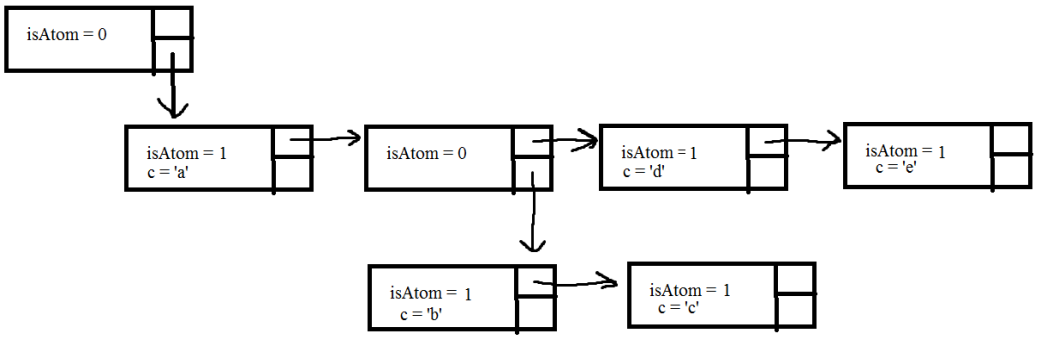

Рисунок 1 - Иллюстрация построения списка

Тестирование.

```
Как будет введена строка?
1. Из файла
2. Вручную
2

Введите строку:
(ab(ab(ab(ab)))cd)
Введенная строка: (ab(ab(ab(ab)))cd)

Введите атом, который нужно найти: a

Построение иерархического списка...
Добавление ответвления
  Добавление в список a
  Добавление в список b
  Добавление ответвления
    Добавление в список a
    Добавление в список b
    Добавление ответвления
      Добавление в список a
      Добавление в список b
      Добавление ответвления
        Добавление в список a
        Добавление в список b
        Завершение добавления ответвления
      Завершение добавления ответвления
    Завершение добавления ответвления
  Добавление в список c
  Добавление в список d
  Завершение добавления ответвления

Вывод элементов списка и поиск в нем атома 'a':
Вызывается рекурсивная функция для дочернего списка
  Проверка атома a - запрашиваемый атом найден!!!
  Проверка атома b - запрашиваемый атом не найден
  Вызывается рекурсивная функция для дочернего списка
    Проверка атома a - запрашиваемый атом найден!!!
    Проверка атома b - запрашиваемый атом не найден
    Вызывается рекурсивная функция для дочернего списка
      Проверка атома a - запрашиваемый атом найден!!!
      Проверка атома b - запрашиваемый атом не найден
      Вызывается рекурсивная функция для дочернего списка
        Проверка атома a - запрашиваемый атом найден!!!
        Проверка атома b - запрашиваемый атом не найден
  Проверка атома c - запрашиваемый атом не найден
  Проверка атома d - запрашиваемый атом не найден

Запрашиваемый элемент был найден!
```

Input	Output
(abc(def()gh)l) j	Запрашиваемый атом не был найден!
a	Запрашиваемый атом был найден!

a	
(a(b)(c)d) c	Запрашиваемый атом был найден!
A(b) b	Введены лишние символы
(B(C(D(E(F(G))))))) A	Введены лишние символы
(a(A(b(B)))) A	Запрашиваемый атом был найден!
(1(c)de) 1	Неверный символ

Вывод.

В ходе выполнения лабораторной работы был усовершенствован навык во владении метода рекурсии. Были изучены основные понятия иерархического списка и реализация его на языке программирования Си.

Приложение.

Код программы.

```
#include <stdio.h>
#include <string.h>
#include <dirent.h>
#include <stdlib.h>
#include <ctype.h>

typedef struct Node{                              //объявление структуры
    char c;                 //символ
    struct Node *next;                //указатель на следующий элемент в
списке
    struct Node *child;                  //указатель на дочерний элемент в
списке
    int isAtom;                       //проверка элемента на атом
}Node;

Node *func(char *s, int count, int *flag);    //главная функция
Node *atom();                              //функция создания атома
void printAndCheckTree(Node *tree, char elem, int *flag, int count);
//функция вывода дерева и нахождения в нем заданного элемента
void indent(int count);            //функция для печати отступов

int curr = -1;

int main(){

    char *file = NULL;            //блок переменных для считывания файлов
    char **files = NULL;
    char string[255];
    FILE *f;
    int textOrType = 0;
    int i = 1;

    char symbol;      //введенный атом, который нужно найти
    int flag = 0;
    Node *list = NULL;


    printf("Как будет введена строка?\n1. Из файла\n2. Вручную\n");
    scanf("%d", &textOrType);
    getchar();
    printf("\n");

    if (textOrType != 1 && textOrType != 2){
        printf("Нужно ввести 1 или 2!\n");
        return 0;
    }
```

```
    else if (textOrType == 2){
        printf("Введите строку:\n");
        fgets(string, 255, stdin);
    }

    else if (textOrType == 1){
        printf("Выберите файл:\n");
        DIR *dir = opendir("directory");
        struct dirent *ent;
        if (dir){
            for (i = 1; (ent = readdir (dir)) != NULL; ){
                if (!strchr(ent->d_name, '.')){
                    files = realloc(files, i * sizeof(char*) + 1);
                    files[i-1]     =     malloc(strlen(ent->d_name)     *
sizeof(char) + 1);
                    strcpy(files[i-1], ent->d_name);
                    printf("%d. %s\n", i, files[i-1]);
                    i++;
                }
            }
        }
        scanf("%d", &textOrType);
        if (textOrType > i-1 || textOrType < 1){
            printf("Неправильно выбран файл!\n");
            return 0;
        }

        file = realloc(file, (strlen(files[textOrType-1]) + 4 + 10) *
sizeof(char) + 1);
        strcpy(file, "directory/");
        strcat(file, files[textOrType-1]);
        file[strlen(file)] = '\0';
        printf("Вы выбрали файл %s\n", files[textOrType-1]);

        for (int j = 0; j < i; j++){
            files[j] = NULL;
            free(files[j]);
        }
        files = NULL;
        free(files);
    }

    f = fopen(file, "r");
    if (f){
        getchar();
        fgets(string, 255, f);
        fclose(f);
        file = NULL;
        free(file);
    }
```

```
    printf("Введенная строка: %s\nВведите атом, который нужно найти: ",
string);
    scanf("%c", &symbol);
    if (symbol == '(' || symbol == ')'){
        printf("Атом введен неверно!\n");
        return 0;
    }

    printf("\n");

    printf("Построение иерархического списка...\n");
    list = func(string, 0, &flag);
    if (string[curr + 1] != '\n'){
        free(list);
        printf("Введены лишние символы!\n");
        return 0;
    }
    if(flag > 0){
        free(list);
        printf("Отсутсвует (\n");
        return 0;
    }
    else if(flag < 0){
        free(list);
        printf("Отсутсвует )\n");
        return 0;
    }

    flag = 0;
    printf("\nВывод элементов списка и поиск в нем атома '%c':\n",
symbol);
    printAndCheckTree(list, symbol, &flag, 0);
    printf("\n");
    if (flag == 1) printf("Запрашиваемый элемент был найден!\n");
    else printf("Запрашиваемый элемент не был найден!\n");

    free(list);
    return 0;
}

Node *func(char *s, int count, int *flag){
    Node *elem = atom();
    curr++;

    if(*flag == 0 && isalpha(s[curr])){
        printf("Добавление в список %c\n", s[curr]);
        elem->c = s[curr];
        elem->isAtom = 1;
        return elem;
```

```
    }

    if(s[curr] == '('){                                                    // при
встрече ( начинает записывать в branch
        indent(count);
        elem->isAtom = 0;
        printf("Добавление ответвления\n");
        (*flag)++;
        elem->child = func(s, count + 1, flag);

        if(elem->child == NULL){
            return NULL;
        }

        if(*flag == 0){
            return elem;
        }

        elem->next = func(s, count, flag);

        if(elem->next == NULL){
            return NULL;
        }
        /*else if(elem->next->isAtom == 0){
            free(elem->next);
            elem->next = NULL;
        }*/

        return elem;
    }

    else if(isalpha(s[curr])){                                              //
при встрече атома продолжает записывать в next

        if(*flag == 0){
            printf("Отсутствует (\n");
            return NULL;
        }

        indent(count);
        printf("Добавление в список %c\n", s[curr]);
        elem->c = s[curr];
        elem->isAtom = 1;
        elem->next = func(s, count, flag);

        if(elem->next == NULL){
            return 0;
        }
        return elem;
    }
```

```
    else if(s[curr] == ')'){                                    //  при
встрече ) завершает запись next
        (*flag)--;
        elem->isAtom = 0;
        indent(count);
        printf("Завершение добавления ответвления\n");
        return elem;
    }

    else if(!isalpha(s[curr])){                                  // при
встрече неверных символов завершает работу
        printf("Неверный сивол\n");
    }

    return NULL;
}

Node *atom(){
    Node *elem = malloc(sizeof(Node));
    elem->child = NULL;
    elem->next = NULL;
    return elem;
}

void printAndCheckTree(Node *tree, char elem, int *flag, int count){
    do{
        if(tree->isAtom){
            indent(count);
            printf("Проверка атома %c - ", tree->c);
            if (tree->c == elem){
                printf("запрашиваемый атом найден!!!\n");
                *flag = 1;
            }
            else printf("запрашиваемый атом не найден\n");
        }
        if(tree->child != NULL){
            indent(count);
            printf("Вызывается   рекурсивная   функция   для   дочернего
списка\n");
            printAndCheckTree(tree->child, elem, flag, count + 1);
        }
        tree = tree->next;
    }while (tree != NULL);
}

void indent(int count){
    for(int i = 0; i < count; i++){
        printf("  ");
    }
}
```