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

Студентка гр. 9381 _____ Москаленко Е.М.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Познакомиться с основными функциями создания и обработки иерархического списка на языке программирования C++.

Задание.

Вариант 10.

Подсчитать число различных атомов в иерархическом списке; сформировать из них линейный список.

Основные теоретические положения.

Согласно рекурсивному определению, иерархический список – это список, элементами которого так же могут быть иерархические списки. Для обработки иерархического списка используются рекурсивные функции, так как он представляет собой множество списков, между которыми установлена иерархия.

На рисунке 1 представлен иерархический список, обрабатываемый созданной программой. Список соответствует сокращенной записи **((a b) c d)**.

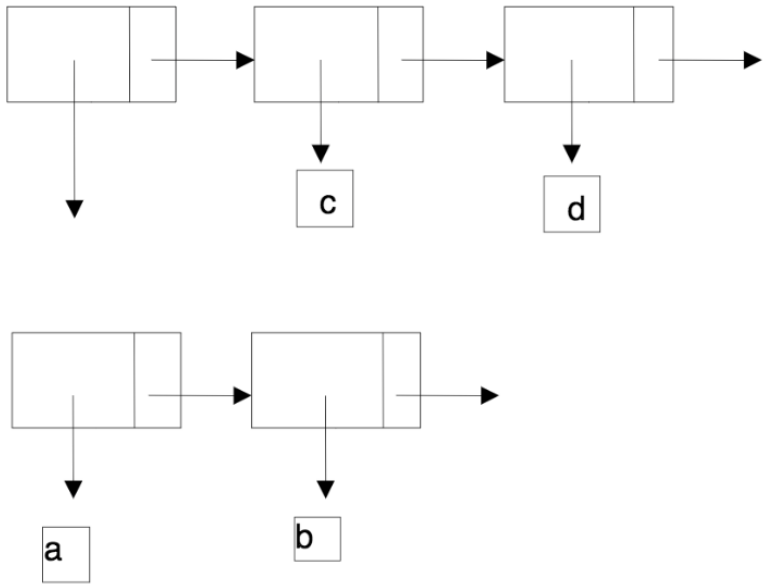

Рисунок 1. **((a b) c d)**.

Ход работы и описание алгоритма.

Алгоритм:

Создаётся указатель на структуру элемента иерархического списка. Затем пользователю предлагается выбрать источник ввода: файл или консоль. Вызывается функция readList для считывания данных (она же вызывает readExp и readRecursion). В этих же функциях создаются атомы с помощью вызова makeAtom и узлы с помощью вызова addNode. Когда список сформирован, то вызывается функция output, которая выводит значение атомов.

Если элемент – узел, то output вызывает рекурсивную функцию outputRecursion, которая выводит голову и хвост узла. Вывод повторяется до конца списка. После этого создается указатель на голову линейного односвязного списка Simple head. В начале значение head = ‘\0’, а следующий элемент не определен.

Вызывается функция makeSimpleList, которая создает линейный список с различными элементами. На вход ей передается указатель на иерархический список и указатель на голову линейного.

Если иерархический список пуст, то происходит выход из функции. Так же в функции инициализируется переменная static int space = 0, отвечающая за пробелы при выводе промежуточных результатов.

Если встретился атом, то он выводится на консоль и происходит проверка значения head линейного списка. Если ее значение = ‘/0’, то вызывается функция initHead и значением head становится атом. Если head уже инициализирована, то вызывается функция checkAtomInSimple и если атома нет в линейном списке, то он добавляется с помощью функции push.

Если же встретился узел, то значение пробела изменяется и рекурсивно вызывается makeSimpleList, в параметры которой передается голова узла. Затем то же самое с хвостом.

После этого с помощью функции listPrint линейный список выводится на консоль, а иерархический список удаляется с помощью функции destroy.

Для работы с иерархическими списками были созданы следующие структуры:

```
struct hlist;
typedef hlist* HListP;
struct Pair {
   HListP head;     - указатель на предыдущий элемент списка
   HListP tail;     - указатель на следующий элемент списка
};
struct hlist {
        int tag;   - переменная, хранящая 1, если элемент списка – атом, 0
        – если узел
        union {
              char atom;
              Pair pair;
           } Node;
};
```

Node – либо значение атома, либо структура Pair с указателями на предыдущий и следующий элементы.

Для работы с иерархическим списком реализованы функции:

int isAtom(HListP list); - проверяет, является ли элемент списка атомом. На вход подается указатель на структуру элемента списка. Возвращаемое значение – 0, если список пустой или элемент – узел, 1 – если атом.

void readList(HListP &list, istream& stream); - считывает пробелы до списка и затем вызывает **readExp.** На вход подается ссылка на список и название потока ввода. Если первый символ после пробелов не (- выводит сообщение об ошибке.

void readExp(char prev, HListP& list, istream& stream); - создает атомы с помощью **makeAtom** и вызывает **readRecursion.** Параметры – последний считанный символ, ссылка на список, название потока ввода. Если первый символ) - выводит сообщение об ошибке.

void readRecursion(HListP& list, istream& stream); - рекурсивная функция, которая обрабатывает строку и создает и скрепляет узлы между собой с помощью вызова addNode. На вход подается ссылка на список и название потока ввода.

void output(HListP list); - выводит на экран список в виде атомов и узлов в виде скобок. На вход подается указатель на структуру списка.

void outputRecursion(HListP list) - выводит на экран список в виде атомов и узлов в виде скобок. Сама функция выводит непосредственно хвост узла. На вход подается указатель на список.

HListP getHead(HListP list) - возвращает указатель на head. На вход подается указатель на структуру списка.

HListP getTail(HListP list) - возвращает указатель на head. На вход подается указатель на структуру списка.

bool isNull(HListP list) – проверяет, пустой ли список. На вход подается указатель на структуру списка. Если список пуст, возвращает false, иначе – true.

HListP addNode(HListP head, HListP tail) - присоединяет узел к списку. На вход подается указатель на голову и хвост узла. Возвращает указатель на присоединенный узел.

HListP makeAtom(const char sign) - создает структуру с атомом. На вход подается значение атом. Возвращает указатель, на созданную структуру.

void makeSimpleList(HListP list, Simple head) – функция создания линейного списка. Аргументы – указатель на структуру списка и указатель на голову линейного списка.

void destroy (HListP s) - уничтожение списка. На вход подается указатель на структуру списка.

Структура линейного списка:

```
struct SimpleList{
   char data;  //значение элемента
   struct SimpleList* next; //ссылка на следующий элемент
};
 typedef SimpleList* Simple;
```

Функции для работы с линейным списком:

void push(Simple head, char data); - добавление элемента в линейный список. Н входе указатель на голову списка и значение нового элемента.

void listPrint(Simple list); - печать линейного списка. На входе указатель на голову.

bool checkAtomInSimple(Simple head, char atom); - проверка вхождения атома в линейный список. На входе указатель на голову и значение атома. Возвращает true, если атом в списке присутствует и false иначе.

void initHead(Simple head, char data); - инициализация головы списка. Параметры – указатель на голову и ее новое значение.

Тестирование.

№	Ввод	Вывод
1	(abc)	Введенный список: (abc) CHECK_HEAD вызов Встретился символ a Создаем НАЧАЛО линейного списка CHECK_HEAD конец CHECK_TAIL вызов CHECK_HEAD вызов Встретился символ b Проверка, есть ли b в линейном списке Добавляем b в линейный список CHECK_HEAD конец CHECK_TAIL вызов CHECK_HEAD вызов

		Встретился символ c Проверка, есть ли c в линейном списке Добавляем c в линейный список CHECK_HEAD конец CHECK_TAIL вызов CHECK_TAIL конец CHECK_TAIL конец CHECK_TAIL конец В линейном списке: a В линейном списке: b В линейном списке: c Количество элементов: 3
2	((abc)dja)	В линейном списке: a В линейном списке: b В линейном списке: c В линейном списке: d В линейном списке: j Количество элементов: 5
3	a b	Элементы списка должны быть в круглых скобках. Попробуйте ещё раз.
4	(a b c (c b a))	В линейном списке: a В линейном списке: b В линейном списке: c Количество элементов: 3
5	)h g 1)	Элементы списка должны быть в круглых скобках. Попробуйте ещё раз.
6	(1(2 3)(56)4(2 1))	В линейном списке: 1 В линейном списке: 2

		В линейном списке: 3 В линейном списке: 5 В линейном списке: 6 В линейном списке: 4 Количество элементов: 6

Выводы.

Был создан иерархический список на языке программирования C++ и освоены рекурсивные функции работы с ним.

ПРИЛОЖЕНИЕ А

ФАЙЛ STRUCTURES.H

```
#ifndef LAB3EXAMPLE_STRUCTURES_H
#define LAB3EXAMPLE_STRUCTURES_H
#include <fstream>
using namespace std;

struct hlist;

typedef hlist* HListP;

struct Pair {
    HListP head; //предыдущий узел
    HListP tail; //следующий
};

struct hlist {
    int tag; //0 - pair, 1 - atom
    union {
        char atom;
        Pair pair;
    } Node;
};

struct SimpleList{
    char data;  //значение
    struct SimpleList* next; //ссылка на следующий элемент
};
 typedef SimpleList* Simple;

int isAtom(HListP list); //проверка, является ли элемент списка атомом
void readList(HListP &list, istream& stream); //считывает пробелы до списка
void readExp(char prev, HListP& list, istream& stream); //создает атомы и вызывает readRecursion
void readRecursion(HListP& list, istream& stream); //Рекурсивная функция, которая обрабатывает строку и создает и скрепляет узлы между собой.
void output(HListP list); //Выводит на экран список в виде атомов и узлов  в виде скобок.
void outputRecursion(HListP list);
HListP getHead(HListP list); //возвращает указатель на head
HListP getTail(HListP list); //возвращает указатель на tail
bool isNull(HListP list); //проверка, пустой ли список
HListP addNode(HListP head, HListP tail); //присоединяет узел к списку
HListP makeAtom(const char sign);
void makeSimpleList(HListP list, Simple head);
void destroy (HListP s); //уничтожение списка

typedef SimpleList* Simple;

void push(Simple head, char data); //добавление элемента в линейный список
void listPrint(Simple list); //печать линейного списка
bool checkAtomInSimple(Simple head, char atom); //проверка вхождения атома в линейный список
void initHead(Simple head, char data); //инициализация головы списка

#endif //LAB3EXAMPLE_STRUCTURES_H
```

ФАЙЛ SIMPLELIST.CPP

```
#include "structures.h"
#include <iostream>
using namespace std;

void push(Simple head, char data){
    cout << "Добавляем " << data << " в линейный список" << "\n";
    Simple current = head;
    while (current->next != nullptr)
        current = current->next;
    Simple node;
    node = new SimpleList;
    node->data = data;
    node->next = nullptr;
    current->next = node;
}

void listPrint(Simple list)
{
    int count = 0;
    Simple p;
    p = list;
    do {
        cout << "\033[32mВ линейном списке:  " << p->data << "\n"; // вывод
значения элемента p
        p = p->next; // переход к следующему узлу
        count++;
    } while (p != nullptr);
    cout << "\033[34m" "Количество элементов: " << count;
}

bool checkAtomInSimple(Simple head, char atom){
    cout << "Проверка, есть ли " << atom << " в линейном списке" << "\n";
    Simple p;
    p = head;
    do {
        if (p->data == atom)
            return true;
        p = p->next; // переход к следующему узлу
    } while (p != nullptr);
    return false;
}

void initHead(Simple head, char data) // data- значение первого узла
{
    cout << "Создаем НАЧАЛО линейного списка" << "\n";
    head->data = data;
}
```

ФАЙЛ MAIN.CPP

```
#include <iostream>
#include "structures.h"
using namespace std;

void printSpace(int space) {
    for (int i = 0; i < space; i++)
        cout << "   ";
}

HListP getHead(HListP list) {
    if (list == nullptr || isAtom(list))  //если список пуст или элемент -
атом
        return nullptr;
    return list->Node.pair.head;
}

HListP getTail(HListP list) {
    if (list == nullptr || isAtom(list))
        return nullptr;
    return list->Node.pair.tail;
}

int isAtom(HListP list) {
    if (list == nullptr)
        return 0;
    return (list->tag);
}

bool isNull(HListP list) {
    return list == nullptr;
}


HListP addNode(HListP head, HListP tail) {
    if (isAtom(tail))
        return nullptr;
    HListP list;
    list = new hlist;
    list->tag = 0;
    list->Node.pair.head = head;
    list->Node.pair.tail = tail;
    return list;
}

HListP makeAtom(const char sign) {
    HListP list;
    list = new hlist;
    list->tag = 1;       //1 - атом
    list->Node.atom = sign; //присваием значение
    return list;
}

void readList(HListP &list, istream& stream) {
    char sign;

    do {
        stream >> sign;
    } while (sign == ' ');
```

```
    if (sign != '('){
        cout << "Элементы списка должны быть в круглых скобках. Попробуйте
ещё раз.";
        exit(1);   //обработка ошибки
    }
    readExp(sign, list, stream);
}

void readExp(char prev, HListP& list, istream& stream) {
    if (prev == ')')
    {
        cerr << "Список не может начинаться с )\n";
        exit(1);
    }
    if (prev != '(')
         list = makeAtom(prev);
    else
        readRecursion(list, stream);
}

void readRecursion(HListP& list, istream& in) {
    char sign;
    HListP p1, p2;
    in >> sign;
    if (sign == ')')
        list = nullptr;
    else {
        readExp(sign, p1, in);
        readRecursion(p2, in);
        list = addNode(p1, p2);         //добавление узла в список
    }
}

void output(HListP list) {
    if (isNull(list))
        cout << "()";

    else if (isAtom(list))
        cout << list->Node.atom;
    else {
        cout << '(';
        outputRecursion(list);
        cout << ')';
    }
}

void outputRecursion(HListP list) {
    if (!isNull(list)) {
        output(getHead(list));
        outputRecursion(getTail(list));
    }
}

void makeSimpleList(HListP list, Simple head) {
    if (isNull(list)) //the end of our list
        return;
    static int space = 0;

    if (isAtom(list)) {
        printSpace(space);
        cout << "\033[34m Встретился символ " << list->Node.atom << "\033[0m"
<< '\n';
```

```
        if (head->data == '\0')
            initHead(head, list->Node.atom);   //инициализация головы
        else if (!checkAtomInSimple(head, list->Node.atom)){
            push(head,list->Node.atom);
        }
    }
    else
        {
        cout << "CHECK_HEAD вызов" << '\n';
        space++;
        printSpace(space);
        makeSimpleList(getHead(list), head);
        space--;
        printSpace(space);
         cout << "CHECK_HEAD конец" << '\n';

        cout << "CHECK_TAIL вызов" << '\n';
        space++;
        printSpace(space);
        makeSimpleList(getTail(list), head);
        space--;
        printSpace(space);
        cout << "CHECK_TAIL конец" << '\n';
    }
}


void destroy (HListP s)
{
    if ( s != nullptr) {
        if (!isAtom(s)) {
            destroy(getHead(s));
            destroy(getTail(s));
        }
        delete s;
    }
}

int main() {
    HListP list = nullptr; //зиация
    string name;
    ifstream file;
    cout << "Выберите:\n1. Ввод списка с консоли\n2. Ввод списка с файла\n";
    int choice = 0;
    cin >> choice;

    switch(choice){
        case 1:
            cout << "Введите список в формате (x y z (f d)). Будьте
внимательнее со скобками.: \n";
            readList(list, cin);
            break;
        case 2:
            cout << "Введите полный путь до файла. В файле не должно быть
строки без закрытых скобок.: \n";
            cin >> name;
            file.open(name);
            if (!file.is_open()){
                cout << "Файл не может быть открыт!\n";
                exit(1);
            }
            readList(list, file);
```

```
            file.close();
            break;
        default:
            cout << "Вы должны ввести 1 или 2";
            return 0;
    }

    cout << "\033[31m Введенный список: \033[0m" << "\n";
    output(list);
    cout << "\n";
    Simple head;
    head = new SimpleList;
    head->next = nullptr;
    head->data = '\0';  //пока head обнулена
    makeSimpleList(list, head);
    listPrint(head);
    destroy(list);
    return 0;
}
```