МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Алгоритмы и структуры данных»

Тема: Иерархические списки

Студент гр. 8304 _____ Перелыгин Д.С.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2019

Цель работы.

Формирование практических навыков работы с иерархическими списками и их рекурсивной обработки. Изучение и практическое применение постфиксной записи арифметических выражений.

Задание.

Пусть выражение (логическое, арифметическое, алгебраическое*) представлено иерархическим списком. В выражение входят константы и переменные, которые являются атомами списка. Операции представляются в префиксной форме ((<операция> <аргументы>)), либо в постфиксной форме (<аргументы> <операция>)). Аргументов может быть 1, 2 и более. Например (в префиксной форме): (+ a (* b (- c))) или (OR a (AND b (NOT c))).

В задании даётся один из следующих вариантов требуемого действия с выражением: проверка синтаксической корректности, упрощение (преобразование), вычисление.Пример упрощения: (+ 0 (* 1 (+ a b))) преобразуется в (+ a b).В задаче вычисления на входе дополнительно задаётся список значений переменных((x1 c1) (x2 c2) … (xk ck)), где xi – переменная, а ci – её значение (константа).В индивидуальном задании указывается: тип выражения (возможно дополнительно - состав операций), вариант действия и форма записи. Всего 9 заданий.

17) логическое, упрощение, префиксная форма

Основные теоретические положения.

Линейный список представляет собой способ организации последовательности однотипных элементов, при котором каждый элемент, кроме первого, имеет одного предшественника (предыдущий элемент) и каждый элемент, кроме последнего, имеет одного преемника (следующий элемент). Доступ к каждому элементу списка можно получить, последовательно продвигаясь по списку от элемента к элементу. Схематично изображённый линейный список представлен на рисунке 1.

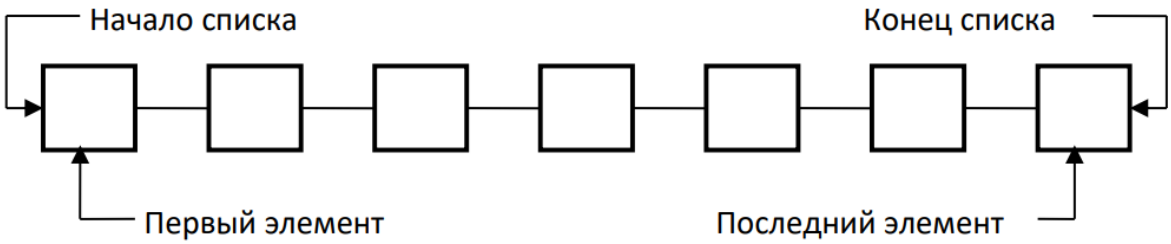

Рис. 1 - Модельное представление списка, обрабатываемого линейно В классической реализации линейные списки обрабатываются последовательно и идёт чёткое определение текущего элемента, в реализации же, использованной в лабораторной работе, применяется подход рекурсивной обработки списка, который логически представлен внутри, как показано на рисунке 2: список делится на отдельный элемент - голову (первый элемент), а все остальные представляется ”хвостом” . Кроме того, использование рекурсивный алгоритмов обусловлено использование не стандартного, а иерархического списка, где каждая функция должна выполняться на отдельном уровне ветвления. Главным отличием данной структуры данных является возможность назначения элемента ссылкой на другой

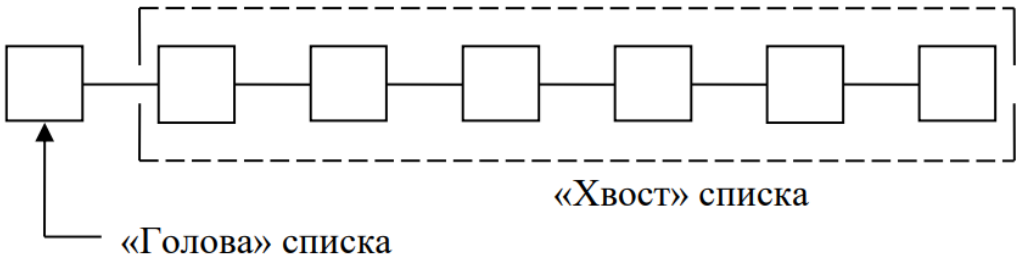

Рис. 2 - Модельное представление списка, обрабатываемого рекурсией иерархический список, за счёт чего и реализуется нелинейность хранения данных и обработки. Схематическое представление списка (a (() (b c) d) e)

представлено на рисунке 3.

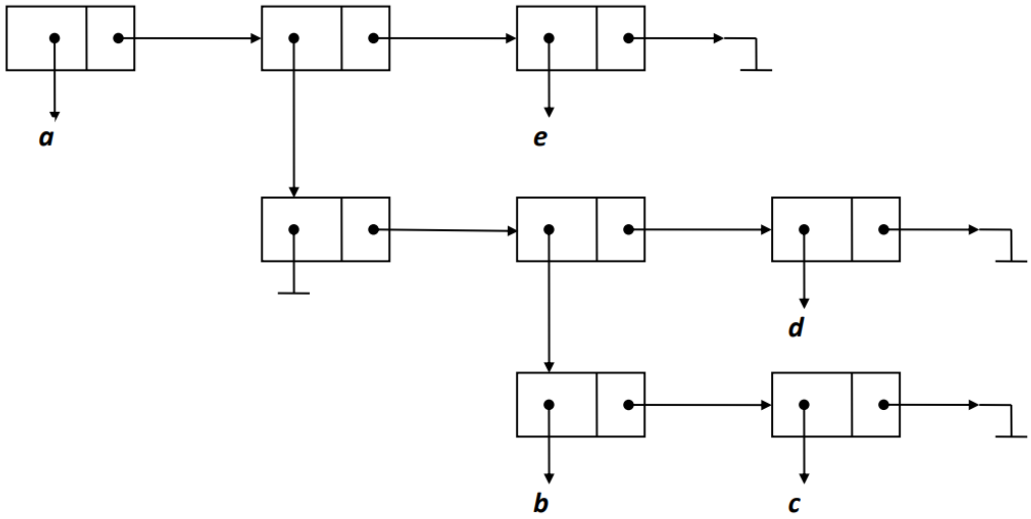

Рис. 3 - Модельное представление списка, обрабатываемого рекурсией

Ход работы.

1. Программа использует следующий алгоритм для выполнения задачи: происходит считывание текста и занесение его в иерархический список. Далее к нему применяется функция red, которая сокращает записанное логическое выражение. Алгоритм спускается вглубь иерархии, доходя до конца, он начинает сворачиваться, запоминая значение переменных, хранящихся в атомах списка. Когда он встречается знак оператора, осуществляется проверка на возможное сокращение из заготовленного списка. Если упрощение возможно, функция записывает его результат на место логического оператора, а затем удаляет элементы, участвующие в сокращении.

2. Написана функция int main() со стандартной сигнатурой, отвечающая за открытие необходимых для ввода-вывода, общение с пользователем и запуск функции-обработчика.

Тестирование.

Входные данные	Ответ программы	Ожидаемый ответ

(&A(|B(-B)))	**(A)**	**(A)**
(&AA)	**(A)**	**(A)**
(&A(&B(-B)))	**(0)**	**(0)**
(&A(&B(C)))	**(&A(&B(C)))**	**(&A(&B(C)))**
(&A(&B(|CC)))	**(&A(&B(C)))**	**(&A(&B(C)))**

Выводы.

В ходе выполнения работы были отработаны навыки работы с иерархическими списками, их представлением, реализацией и обработкой. Создана программа для сокращения несложных логический выражений с необходимым функционалом.

Приложение А.

Файл lisp.h

```
#pragma once
namespace h_list {
	typedef char base;

	struct s_expr;
	struct two_ptr;

	struct two_ptr {
		s_expr* hd;
		s_expr* tl;
	};

	struct s_expr {
		bool tag;
		union {
			base atom;
			two_ptr pair;
		}node;
	};

	typedef s_expr* lisp;

	lisp head(const lisp s);
	lisp tail(const lisp s);
	lisp cons(const lisp h, const lisp t);
	lisp make_atom(const base x);
	void read_lisp(lisp& y);
	void read_s_expr(base prev, lisp& y);
	void read_seq(lisp& y);
	void write_lisp(const lisp x);
	void write_seq(const lisp x);
	bool isNull(const lisp s);
	bool isAtom(const lisp s);
	void destroy(lisp s);
	void red(lisp s, int& flag, char& a, char& b);
```

```
}
```

Файл Source.cpp

```
#include <iostream>
#include <cstdlib>
#include "lisp.h"
#include <clocale>
#include <windows.h>
using namespace h_list;
using namespace std;


int main() {
	SetConsoleCP(1251);
	SetConsoleOutputCP(1251);
	setlocale(LC_ALL, "Russian");
	lisp s;
	char a = '9', b = '9';
	int flag = 0;
	cout << boolalpha;
	cout << "ВВедите список:    ";
	read_lisp(s);

	red(s, flag, a, b);
	cout << a << endl;
	write_lisp(s);
	cout << endl;

	return 0;
	}
```

Файл Function.cpp

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

namespace h_list {

	lisp head(const lisp s) {
		if (s != NULL)
			if (!isAtom(s))
				return s->node.pair.hd;
			else { cerr << "Error: Head(atom) \n"; exit(1); }
		else {
			cerr << "Error: Head(nil) \n";
			exit(1);
		}
	}

	bool isAtom(const lisp s) {
		if (s == NULL)
			return false;
		else return (s->tag);
	}

	bool isNull(const lisp s) {
		return s == NULL;
	}

	lisp tail(const lisp s) {
```

```
	if (s != NULL)
		if (!isAtom(s))
			return s->node.pair.tl;
		else {
			cerr << "Error: Tail(atom) \n";
			exit(1);
		}
	else {
		cerr << "Error: Tail(nil) \n";
		exit(1);
	}
}

lisp cons(const lisp h, const lisp t) {
	lisp p;
	if (isAtom(t)) {
		cerr << "Error: Cons(*, atom)\n";
		exit(1);
	}
	else {
		p = new s_expr;
		if (p == NULL) {
			cerr << "Memory not enough\n";
			exit(1);
		}
		else {
			p->tag = false;
			p->node.pair.hd = h;
			p->node.pair.tl = t;
			return p;
		}
	}
}

lisp make_atom(const base x) {
	lisp s;
	s = new s_expr;
	s->tag = true;
	s->node.atom = x;
	return s;
}

base getAtom(const lisp s) {
	if (!isAtom(s)) {
		//cerr << "Error: getAtom(s) for !isAtom(s)" << endl;
		//exit(1);
		return '0';
	}
	else return (s->node.atom);
}

void read_lisp(lisp& y)
{
	base x;
	do {
		cin >> x;
	} while (x == ' ');
	read_s_expr(x, y);
}

void read_s_expr(base x, lisp& y) {
	if (x == ')') {
		cerr << "Error:Missing '('" << endl;
		exit(1);
```

```
	}
	else if (x != '(')
		y = make_atom(x);
	else {
		char buffer = cin.peek();
		if (buffer == ')') {
			cout << "Error: List emtry!" << endl;
			exit(1);
		}
		read_seq(y);
	}
}


void read_seq(lisp& y) {
	base x;
	lisp p1, p2;
	if (!(cin >> x)) {
		cerr << " ! List.Error 2 " << endl;
		exit(1);
	}
	else {
		while (x == ' ')
			cin >> x;
		if (x == ')')
			y = NULL;
		else {
			read_s_expr(x, p1);
			read_seq(p2);
			y = cons(p1, p2);
		}
	}
}

void write_lisp(const lisp x) {
	if (isNull(x))
		cout << " ()";
	else if (isAtom(x))
		cout << ' ' << x->node.atom;
	else {
		cout << " (";
		write_seq(x);
		cout << " )";
	}
}

void write_seq(const lisp x) {
	if (!isNull(x)) {
		write_lisp(head(x));
		write_seq(tail(x));
	}
}

void destroy(lisp s)
{
	if (s != NULL) {
		if (!isAtom(s)) {
			destroy(head(s));
			destroy(tail(s));
		}
		delete s;
		// s = NULL;
	}
}
```

```
void red(lisp s, int& flag, char& a, char& b)
{
	//cout << getAtom(s) << endl;
	if (isNull(tail(s)) && (isAtom(head(s))))
	{
		cout << "back" << endl;
		a = getAtom(head(s));
		cout << a << endl;
		return;
	}
	if (!(isAtom(head(s))) && head(s) != NULL)
	{
		cout << "head"  << endl;
		red(head(s), flag, a, b);
	}
	else if (!(isAtom(tail(s))) && tail(s) != NULL)
	{
		cout << "tail"  << endl;
		red(tail(s), flag, a, b);
	}
	if (isAtom(head(s)))
	{
		cout << "up" << '\n' << a << '\n' << b  << endl ;

		if (getAtom(head(s)) == '-')
		{
			flag = 1;
			return;
		}

		if (getAtom(head(s)) == '&')
		{
			cout << "AND" << endl;
			if ((a == b) && flag)
			{
				cout << "1" << endl;
				head(s)->node.atom = '0';
				a = getAtom(head(s));
				b = '9';
				flag = 0;
				cout << "DESTROY" << endl;
				destroy(tail(s));
				s->node.pair.tl = NULL;
				return;
			}

			if ((a == b) && !flag)
			{
				cout << "2" << endl;
				head(s)->node.atom = a;
				a = getAtom(head(s));
				b = '9';
				flag = 0;
				cout << "DESTROY" << endl;
				destroy(tail(s));
				s->node.pair.tl = NULL;
				return;
			}

			if ((a == '1' || b == '1') && !flag)
			{
				if (a=='1')
```

```
                {
                        head(s)->node.atom = b;
                        a = getAtom(head(s));
                }
                else
                {
                        head(s)->node.atom = a;
                        a = getAtom(head(s));
                }
                b = '9';
                flag = 0;
                cout << "DESTROY" << endl;
                destroy(tail(s));
                s->node.pair.tl = NULL;
                return;
        }

        if ((a == '0' || b == '0') && !flag)
        {
                        head(s)->node.atom = '0';
                        a = getAtom(head(s));
                b = '9';
                flag = 0;
                cout << "DESTROY" << endl;
                destroy(tail(s));
                s->node.pair.tl = NULL;
                return;
        }

}

if (getAtom(head(s)) == '|')
{
        cout << "OR" << endl;
        if ((a == b) && flag)
        {
                cout << getAtom(head(s)) << endl;
                head(s)->node.atom = '1';
                a = getAtom(head(s));
                b = '9';
                flag = 0;
                cout << "DESTROY" << endl;
                destroy(tail(s));
                s->node.pair.tl = NULL;
                return;
        }

        if ((a == b) && !flag)
        {
                head(s)->node.atom = a;
                a = getAtom(head(s));
                b = '9';
                flag = 0;
                cout << "DESTROY" << endl;
                destroy(tail(s));
                s->node.pair.tl = NULL;
                return;
        }

        if ((a == '1' || b == '1') && !flag)
        {
                head(s)->node.atom = '1';
                a = getAtom(head(s));
                b = '9';
```

```
                        flag = 0;
                        cout << "DESTROY" << endl;
                        destroy(tail(s));
                        s->node.pair.tl = NULL;
                        return;
                }

                if ((a == '0' || b == '0') && !flag)
                {
                        if (a == '0')
                        {
                                head(s)->node.atom = b;
                                a = getAtom(head(s));
                        }
                        else
                        {
                                head(s)->node.atom = a;
                                a = getAtom(head(s));
                        }
                        b = '9';
                        flag = 0;
                        cout << "DESTROY" << endl;
                        destroy(tail(s));
                        s->node.pair.tl = NULL;
                        return;
                }

        }

        else
        {
                if (a == 9)
                        a = getAtom(head(s));
                else
                        b = getAtom(head(s));

        }

}
else
{
        return;
}
}
}
```