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. 9303 _____ Отмахова М.А.

Преподаватель _____ Филатов Ар.Ю.

Санкт-Петербург

2020

Цель работы.

Ознакомиться с понятием «иерархический список», реализовать программу, решающая данную задачу на языке C++.

Задание.

Вариант 18.

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

((x 1 c 1) (x 2 c 2) … (x k c k)),

где x i – переменная, а c i – её значение (константа).

18) логическое, вычисление, добавить 4-ую операцию (которая может принимать 2 аргумента), префиксная форма

Основные теоретические положения.

Основные теоретические положения.

Иерархический список согласно определению представляет собой или элемент базового типа El, называемый в этом случае атомом (атомарным S-выражением), или линейный список из S-выражений. Приведенное определение задает структуру непустого иерархического списка как элемента размеченного объединения [1] множества атомов и множества пар «голова»–«хвост» и порождает различные формы представления в зависимости от принятой формы представления линейного списка. Традиционно иерархические списки представляют или графически, используя для изображения структуры списка двухмерный рисунок, или в виде одномерной скобочной записи.

Выполнение работы

Функция main

После запуска, программа запрашивает пользователя ввести сначала строку значения переменных, далее – выражение, которое нужно вычислить.

Программа выводит результат, вызывая функцию counter, которая высчитывает ответ. В функцию мы передаем наш список, предварительно убрав скобки внутри с помощью функции flatten1 и перевернув список с помощью функции reverse, также в функцию counter передаем строку значений.

Основная функция, обрабатывающая входные данные – функция analyze.

В зависимости от последней буквы операции (AND, OR, NOT, ORN), функция применяет логические операции к аргументам.

Функция, в которой происходит вызов основной вычисляющей функции – функция counter. Она в цикле while вызывает функцию analyze, пока список не окажется пустым.

Выводы.

В ходе выполнения лабораторной работы была создана программа, принимающая на вход список переменных и их значений, логическое выражение и считает значение данного логического выражения.

Также было изучено понятие «иерархический список» и принципы работы с нним.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main.cpp

```
#include <iostream>
#include "l_intrfc.h"
#include <string>

using namespace std;
using namespace h_list;

lisp concat (const lisp y, const lisp z);
bool counter(lisp, string);
lisp reverse(const lisp s);
lisp rev(const lisp s,const lisp z);

lisp flatten1(const lisp s);

int main ( )
{
    lisp s1;
    string values;
    cout << "enter values:" << endl;
    getline(cin,values);
```

```
    cout << "enter list:" << endl;
    read_lisp (s1);
    cout << "result: " << endl;
    cout << counter(reverse(flatten1(s1)), values) <<
endl;
    destroy (s1);
    return 0;
}

lisp concat (const lisp y, const lisp z)
{
    if (isNull(y)) return copy_lisp(z);
    else return cons (copy_lisp(head (y)), concat (tail
(y), z));
}

lisp reverse(const lisp s)
{
    return(rev(s,NULL));
}


lisp rev(const lisp s,const lisp z)
{
    if(isNull(s)) return(z);
    else if(isAtom(head(s))) return(rev(tail(s),
cons(head(s),z)));
    else return(rev(tail(s), cons(rev(head(s),
NULL),z)));
}
```

```
lisp flatten1(const lisp s)
{
    if (isNull(s)) return NULL;
    else if(isAtom(s)) return
cons(make_atom(getAtom(s)),NULL);
    else
    if (isAtom(head(s))) return cons(
make_atom(getAtom(head(s))),flatten1(tail(s)));
    else
        return
concat(flatten1(head(s)),flatten1(tail(s)));
}


bool analyze(lisp &list, string values, bool answer){
    int pos = values.find(head(list)->node.atom);
    list = tail(list);
    bool answer1;
    if(values[pos+2] == '1')
        answer1 = true;
    else
        answer1 = false;
    switch (head(list)->node.atom) {
        case 'R':
            answer = answer || answer1;
            list = tail(list);
            list = tail(list);
            return answer;
```

```
            break;
        case 'D':
            answer = answer && answer1;
            list = tail(list);
            list = tail(list);
            list = tail(list);
            return answer;
            break;
        case 'T':
            answer = !answer1;
            list = tail(list);
            list = tail(list);
            list = tail(list);
            return answer;
            break;
        case 'N':
            answer = !(answer || answer1);
            list = tail(list);
            list = tail(list);
            list = tail(list);
            return answer;
            break;
    }
}

bool counter(lisp list, string values) {
    bool answer;
    while (!isNull(list)) {
        answer = analyze(list, values, answer);
    }
```

```
    return answer;
}
```

Файл l_impl.cpp

```
#include "l_intrfc.h"
#include <iostream>


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)
    {// PreCondition: not null (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: Tail(nil) \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;
}


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

base getAtom (const lisp s)
{
     if (!isAtom(s)) { cerr << "Error: getAtom(s) for
!isAtom(s) \n"; exit(1);}
     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 prev, lisp& y)
    {
         if ( prev == ')' ) {cerr << " ! List.Error 1 " <<
endl; exit(1); }
         else if ( prev != '(' ) y = make_atom (prev);
         else 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));
        }
    }

    lisp copy_lisp (const lisp x)
    {    if  (isNull(x)) return NULL;
        else if (isAtom(x)) return make_atom (x-
>node.atom);
        else return cons (copy_lisp (head (x)), copy_lisp
(tail(x)));
    }

}
```

Файл l_intrfc.h

```
namespace h_list

{
    typedef char base;

    struct s_expr;

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

    struct s_expr {
        bool tag; // true: atom, false: pair
        union
        {
            base atom;
            two_ptr pair;
        } node;
    };

    typedef s_expr *lisp;


    void print_s_expr( lisp s );

    lisp head (const lisp s);
    lisp tail (const lisp s);
    lisp cons (const lisp h, const lisp t);
```

```
    lisp make_atom (const base x);
    bool isAtom (const lisp s);
    bool isNull (const lisp s);
    void destroy (lisp s);

    base getAtom (const lisp s);

    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);

    lisp copy_lisp (const lisp x);

}
```

ПРИЛОЖЕНИЕ В
ТЕСТИРОВАНИЕ

Таблица 1 - Тестирование программы

Входные данные	Результат работы программы
enter values: ((a 1)(b 0)(c 0)) enter list: (AND a(OR b(NOT c)))	result: 1
enter values: (a 0)(b 1) enter list: (AND a(NOT b))	result: 0
enter values: (a 0)	result: 1
enter values: (a 0)(b 1)(c 0)(d 1) enter list: (ORN a(AND b(OR c(NOT b))))	result: 1
enter values: ((a 0)(b 0)(c 0)) enter list: (ORN a(OR b(NOT c)))	result: 0