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

Студентка гр. 7381 _____ Алясова А.Н.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2018

Задание.

Вариант 10.

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

Описание алгоритма.

На вход программе подается строка в виде иерархического списка. Программа обрабатывает строку, подсчитывает количество различных атомов, если они есть, и выводит линейный список различных атомов и их количество. В противном случае — будет выведено сообщение о том, что список пустой. Если на вход программе подать линейный список, то будет считан лишь 1 элемент строки, остальные будут проигнорированы. В таком случае список состоит из 1 атома.

Описание функций.

void print_s_expr(lisp s);

lisp head (const lisp s); - возвращает указатель на голову

lisp tail (const lisp s); - возвращает указатель на хвост

lisp cons (const lisp h, const lisp t); - конструирует элемент

lisp make_atom (const base x); - создание атома

bool isAtom (const lisp s); - проверка на атом

bool isNull (const lisp s); - проверка на пустоту

void destroy (lisp s); - очищение списка

base getAtom (const lisp s); - получение атома

Функции ввода:

void read_lisp (lisp &y);

void read_s_expr (base prev, lisp &y);

void read_seq (lisp &y);

Функции вывода:

void write_lisp (const lisp x, std::list<char> &mList);

void write_seq (const lisp x, std::list<char> &mList);

void print_list(list<char> &mList); - вывод списка аргументов в консоль

Тестирование.

На вход программе подается строка в виде иерархического списка, на выходе – строка в виде линейного списка различных атомов и количество различных атомов. Приведем примеры в табл. 1. Для автоматизации тестирования, был написан bash-скрипт.

Таблица 1 – Тестирование

Входные данные	Результат
(dwqd(ttr(uy))pop)	dopqrtuwy 9
(ddd(uyt)()(d)jkk)	djktuy 6
((()))	List is empty, cause hierarchial list has no atoms
l	l 1
((asksl)(fgdy())	List.Error 2
(ytrwbn)	bnrtwy 6

Вывод.

В процессе выполнения лабораторной работы были получены навыки по обработке иерархических списков, написанию bash-скриптов и автоматизации тестирования. Работа была написана на языке программирования C++.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

main.cpp

```
#include <iostream>
#include <cstdlib>
#include "function.h"

using namespace std;
using namespace h_list;

void print_list(list<char> &mList);

int main (){
    cout << "Enter your hierarchial list: \n";
    lisp s1;
    list<char> mList;
    cout << "Linear list after processing:  ";
    read_lisp (s1);

    write_lisp (s1, mList);
    mList.sort();
    mList.unique();

    print_list(mList);
    destroy (s1);
    return 0;
}

//============ВЫВОД СПИСКА АРГУМЕНТОВ В КОНСОЛЬ=============//
void print_list(list<char> &mList){
    unsigned long i = 0;
    if( mList.empty() ) {
```

```
     cout << "List is empty, cause hierarchial list has no atoms.\n";
     return;
     }
    list<char> :: iterator it;
    for (it = mList.begin(); it != mList.end(); it++, i++){
     if(i == mList.size()-1) {
                cout << (*it);
                break;
     }
     cout << (*it) << " <-> ";
}
    cout << '\n' << "Count of different elements [" << i+1 << "]\n";
}
```

function.h

```
    #include <iostream>
    #include <list>
    #include <cstdlib>

    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);

    // функции ввода:
    void read_lisp ( lisp &y);            // основная
    void read_s_expr (base prev, lisp &y);
    void read_seq ( lisp &y);

    // функции вывода:
    void write_lisp (const lisp x, std::list<char> &mList);          //
основная
    void write_seq (const lisp x, std::list<char> &mList);

}
```

function.cpp

```
#include "function.h"
#include <iostream>
#include <cstdlib>
```

```
using namespace std;
namespace h_list{

//==============ВОЗВРАЩАЕТ УКАЗАТЕЛЬ НА ГОЛОВУ================//
 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);
      }
 }
//============================================================//

//====================ПРОВЕРЯЕТ АТОМ ЛИ ЭТО====================//
 bool isAtom (const lisp s){
      if(s == NULL)
            return false;
      else
            return (s -> tag);
 }
//============================================================//

//================ПРОВЕРКА НА ПУСТОТУ СПИСКА==================//
 bool isNull (const lisp s){
      return s == NULL;
 }
```

```
//============================================================//

//=============ВОЗВРАЩАЕТ УКАЗАТЕЛЬ НА ХВОСТ==================//
 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);
      }
 }
//============================================================//

//==================КОНСТРУИРУЕТ ЭЛЕМЕНТ======================//
 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;          //выделение памяти
           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;
           }
      }
 }
//============================================================//

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

//=================ОЧИЩЕНИЕ СПИСКА===========================//
 void destroy (lisp s){
      if ( s != NULL){
           if (!isAtom(s)){
                destroy ( head (s) );
                destroy ( tail(s) );
           }
      delete s;                          //на выходе s = NULL;
      }
 }
//============================================================//

//================ПОЛУЧЕНИЕ АТОМА===========================//
```

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

//======================ЧТЕНИЕ=============================//
 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);
           }
      }
 }
//============================================================//

//============Запись в список атомов=======================//
 void write_lisp (const lisp x, list<char> &mList){
      if (isAtom(x))
           mList.push_back(x->node.atom);
      else  {
           write_seq(x, mList);
      }
 }
//============================================================//
```

```
    //=============================================================//
     void write_seq (const lisp x, list<char> &mList){
          if (!isNull(x)) {
               write_lisp(head (x),mList);
               write_seq(tail (x), mList);
          }
     }
    //=============================================================//
}
```