МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по лабораторной работе №2
по дисциплине «Алгоритмы и структуры данных»
ТЕМА: ИЕРАРХИЧЕСКИЕ СПИСКИ

Студентка гр. 7381 _____ Кушкоева А.О.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2018

1. ЦЕЛЬ РАБОТЫ

Цель работы: познакомиться с иерархическими списками и использованием их в практических задачах на языке программирования C++.

Формулировка задачи: сформировать линейный список атомов исходного иерархического списка таким образом, что скобочная запись полученного линейного списка будет совпадать с сокращенной скобочной записью исходного иерархического списка после устранения всех внутренних скобок.

2. РЕАЛИЗАЦИЯ ЗАДАЧИ

В функции main было реализовано меню для пользователя, где можно выбрать способ ввода входных данных. Данные можно ввести либо из файла, либо из терминала. Базовым типом данных для данной задачи является тип char. Для реализации иерархического списка использовались две структуры: struct s_expr и struct two_ptr. Структура struct two_ptr содержит в себе два указателя s_expr *hd и s_expr *tl. Структура struct s_expr содержит переменную bool tag, которая в зависимости от того, является элемент атомом или подсписком списка присваивает значение true и false соответственно. Также эта структура содержит объединение двух типов, base atom и two_ptr pair.

Функции-селекторы: head и tail, выделяющие «голову» и «хвост» списка соответственно. Если «голова» списка не атом, то функция head возвращает список, на который указывает голова пары, т.е. подсписок, находящийся на следующем уровне иерархии. Если же «голова» списка — атом, то выводится сообщение об ошибке и функция прекращает работу. Функция tail работает аналогично функции head, но только для «хвоста».

Функции-конструкторы: Cons, создающая точечную пару (новый список из «головы» и «хвоста»), и Make_Atom, создающая атомарное выражение. При создании нового выражения требуется выделение памяти. Если памяти нет, то p == NULL и это приводит к выводу соответствующего сообщения об ошибке. Если «хвост» — не атом, то для его присоединения к «голове» требуется создать новый узел (элемент), головная ссылка которого

будет ссылкой на «голову» этого «хвоста», а хвостовая часть элемента (tag.hd.tl) ─ ссылкой на его «хвост»

Функции-предикаты: isNull, проверяющая список на отсутствие в нем элементов, и Atom, проверяющая, является ли список атомом. Если элемент – атом, тогда функция возвращает значение tag, которое равно true, и значение False, если «голова-хвост». В случае пустого списка значение предиката False.

Функции-деструкторы: delete и destroy. Функция delete удаляет текущий элемент из списка, а функция destroy удаляет весь список путем вызова функции delete и вызова самой себя.

Функция getAtom возвращает нам значение атома.

Функция copy_list – функция копирования списка.

Функция concat – функция для соединения двух списков. Создает новый иерархический список из копий атомов, входящих в соединяемые списки.

Функция flatten – функция для выравнивания иерархического списка, то есть формирования из него линейного списка путем удаления из сокращенной скобочной записи иерархического списка всех внутренних скобок. Для ввода и вывода иерархического списка были написаны отдельные функции в соответствии с сокращенной скобочной записью иерархического списка.

3. ТЕСТИРОВАНИЕ

Программа была собрана в компиляторе g++ с ключом -std=c++14 в OS Linux Ubuntu 16.04 LTS.

В ходе тестирования ошибки не были найдены.

Некорректные случаи представлены в табл. 1, где была написана неправильная форма записи иерархического списка.

Таблица 1 — Некорректные случаи в синтаксисе.

Входные данные	Результат
)dqwqd	! List.Error 1 — нет открывающей скобки
(dqwq(dqd)	! List.Error 2 - нет закрывающей скобки

Корректные тестовые случаи представлены в приложении А.

4. ВЫВОД

В ходе работы были получены навыки работы с иерархическими списками. Поскольку структура иерархического списка определена рекурсивно, рекурсивный подход является простым и удобным способом поиска решения.

ПРИЛОЖЕНИЕ А.
ТЕСТОВЫЕ СЛУЧАИ

Таблица 2 — Корректные тестовые случаи

Входные данные	Результат
(a (f f g d (f w e f w r (q e w r))))	(a f f g d f w e f w r q e w r)
(q w d q (d q w d q d q w (D q w q d q)))	(q w d q d q w d q d q w D q w q d q)
()	()
(((((a)))))	(a)

ПРИЛОЖЕНИЕ Б.
ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <iostream>
#include <sstream>
#include <cstdlib>
#include <fstream>
#include <cstring>
using namespace std;

typedef char base; // базовый тип элементов (атомов)
struct s_expr; struct two_ptr{     s_expr *hd;
s_expr *tl; }; //end two_ptr; struct s_expr {
    bool tag; // true: atom, false: pair
    union{
base atom;
two_ptr pair;
    }node; //end union node
}; //end s_expr
typedef s_expr *lisp;
// функции
void print_s_expr( lisp s );
void syntax( base x, lisp& y);
// базовые функции:
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, istream
&is_str); // основная void read_s_expr (base prev,
lisp& y, istream &is_str); void read_seq ( lisp& y,
istream &is_str);
// функции вывода: void write_lisp (const
lisp x); // основная void write_seq (const
lisp x); lisp copy_lisp (const lisp x);
lisp concat (const lisp y, const lisp z);
lisp flatten(const lisp s);
//.................................     lisp head
(const lisp s){// PreCondition: not null (s)         if (s
!= nullptr)             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
== nullptr)             return false;
else
```

```
            return (s -> tag);
}
    //.....................................
bool isNull (const lisp s){         return
s==nullptr;     }
    //.....................................     lisp
tail (const lisp s){// PreCondition: not null (s)
        if (s != nullptr)
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){
    // PreCondition: not isAtom (t)
lisp p;         if (isAtom(t)) {
            cerr << "Error: Cons(*, atom)\n";
exit(1);
        }         else {             p = new
s_expr;             if ( p == nullptr) {
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 != nullptr) {             if
(!isAtom(s)){
destroy ( head (s));
destroy ( tail(s));
            }
delete s;
//s = NULL;
        }
    }
    //.........................
base getAtom (const lisp s){
if (!isAtom(s)) {
            cerr << "Error: getAtom(s) for !isAtom(s) \n";
exit(1);
```

```
        }
else
            return (s->node.atom);
}
//.........................
    // ввод списка с консоли
    void read_lisp ( lisp& y, istream &is_str){
base x;         do{             is_str >> x;
}while (x==' ');
        read_s_expr ( x, y, is_str);
} //end read_lisp
    //.........................     void read_s_expr (base prev, lisp&
y, istream &is_str){ //prev – ранее прочитанный символ}         if ( prev
== ')' ) {
            cerr << " ! List.Error 1 - нет открывающей скобки" << endl;
exit(1);
        }
        else if ( prev != '(' )
            y = make_atom (prev);
else
            read_seq (y, is_str);
} //end read_s_expr
    //.........................     void
read_seq ( lisp& y, istream &is_str){
        base x;         lisp
p1, p2;         if (!(is_str
>> x)) {
            cerr << " ! List.Error 2 - нет закрывающей скобки " << endl;
            exit(1);
        }
else {
            while ( x==' ' ){
is_str >> x;             }

            if ( x == ')' )
y = nullptr;
else {
                read_s_expr ( x, p1, is_str);
read_seq ( p2, is_str);                 y =
cons (p1, p2);
            }
        }
    } //end read_seq
//.........................

    // Процедура вывода списка с обрамляющими его скобками – write_lisp,

    // а без обрамляющих скобок – write_seq
    void write_lisp (const lisp x){//пустой список выводится как ()
        if (isNull(x))             cout
<< " ( )";         else if (isAtom(x))
cout << ' ' << x->node.atom;
else { //непустой список}
cout << " (" ;             write_seq(x);
cout << " )";
        }
    } // end write_lisp
    //.........................     void write_seq (const lisp
x){//выводит последовательность элементов
```

```
списка без обрамляющих его скобок
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)));
}

    lisp flatten(const lisp s){
if (isNull(s))
return NULL;
        else if(isAtom(s))
            return cons(make_atom(getAtom(s)),nullptr);
        else if (isAtom(head(s)))
            return cons( make_atom(getAtom(head(s))),flatten(tail(s)));
        else //Not Atom(Head(s))
            return concat(flatten(head(s)),flatten(tail(s)));
}

    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));
    } // end concat
    // -----------------------
 //end copy-lisp

int main ( )
{
    lisp s1, s2;
filebuf file;
string file_name;
stringbuf exp;
string temp_str;
char c;
   // ifstream infile ("a.txt");
    int run = 1;
int k;
while(run){
        cout<<"Введите 1, если хотите ввести выражение из консоли, введите 2,
если хотите ввести выражение из файла, 3 - выход из программы."<<endl;
cin>>k;         cin.ignore();         switch(k){             case 1:{
                cout << "введите list1:" << endl;
                getline(cin, temp_str);
istream is_str(&exp);
exp.str(temp_str);                  read_lisp (s1,
is_str)                 cout << "введен list1: "
<< endl;                 write_lisp (s1);
cout << endl;
                cout << "flatten списка = " << endl;
```

```
                s2 = flatten (s1);
write_lisp (s2);
cout << endl;
                cout << "destroy list1, list2: " << endl;
                destroy (s2);
destroy(s1);
            break;
            }
case 2:{
                    ifstream infile("a.txt");
getline(infile, temp_str);
istream is(&exp);
exp.str(temp_str);
read_lisp (s1, is);
                    cout << "введен list1: " << endl;
write_lisp (s1);                    cout << endl;
                    cout << "flatten списка = " << endl;
                    s2 = flatten (s1);
write_lisp (s2);
cout << endl;
                    cout << "destroy list1, list2: " << endl;
                    destroy (s2);
destroy(s1);                 break;
}             case 3:
cout<<"End!"<<endl;
return 0;             default:
break;         }

    }
return 0;
}
```