ММИНОБРНАУКИ 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: Рекурсивная обработка иерархических списков.

Студент гр. 9384 Звега А.Р.

Преподаватель Ефремов М.А.

Санкт-Петербург
2020

Цель работы.

Создать линейный список из иерархического.

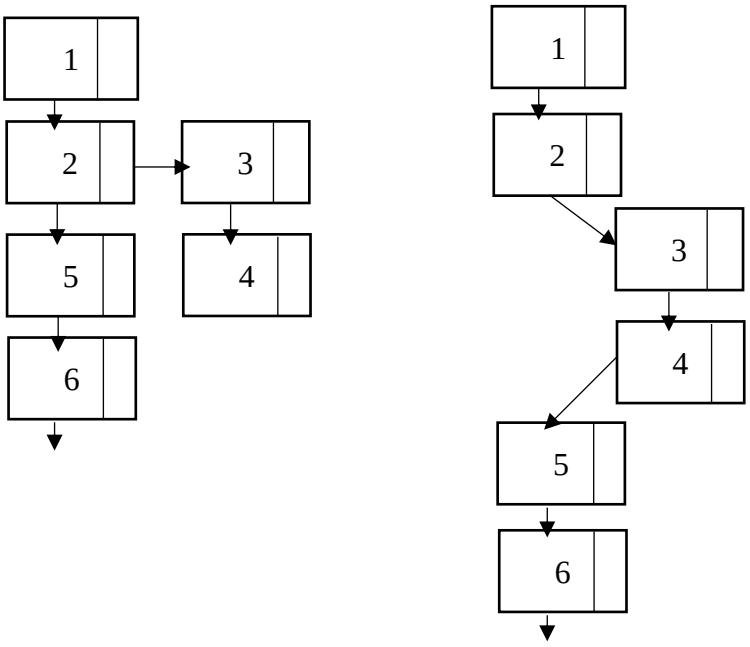
Задание.

ВАРИАНТ 6.

6. сформировать линейный список атомов исходного иерархического списка путем устранения всех внутренних скобок в его сокращенной скобочной записи;

Выполнение работы.

Чтобы из иерархического списка создать линейный, нам нужно чтобы все хвосты указывали на NULL, а головы указывали на хвосты (если они не равны NULL) и голова последнего элемента хвоста должна указывать на следующий элемент списка:



Таким образом получается классический линейный список, в котором каждый элемент имеет указатель на следующий, отсутствует какое-либо ветвление.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

Входные данные	Выходные данные
(1 2 (3 4) 5 6)	(1 2 3 4 5 6)
(a b (s d (g h) a (d) c) b)	(a b s d g h a d c b)
(a b ((s d (g h) a (d) c) b a (k) d k a g))	(a b s d g h a d c b a k d k a g)

Выводы.

В ходе выполнения лабораторный работы был создан линейный список из иерархического. Так же были получены навыки работы с иерархическим списком.

ПРИЛОЖЕНИЕ
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

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

using namespace std;
using namespace h_list;
lisp lineList(const lisp s);
lisp lineRest(const lisp h, lisp t);

int main()
{
    lisp s1, s2;
    cout << boolalpha;
    cout << "введите list:" << endl;
    read_lisp(s1);
    cout << "введен list: " << endl;
    write_lisp(s1);
    cout << endl;
    s2 = lineList(s1);
    cout << "для списка:" << endl;
    write_lisp(s1);
    cout << endl;
    cout << "раскрытый список есть:" << endl;
    write_lisp(s2);
    cout << endl;
    cout << "end! " << endl;
    return 0;
}

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

lisp lineList(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))), lineList(tail(s)));
        else
            return lineRest(lineList(head(s)), lineList(tail(s)));
    }
```

Название файла: lisp.h

```
namespace h_list{
    typedef char base;

    struct exp;
    struct ptrs{
        exp *head;
        exp *tail;
    };
    struct exp{
        bool tag;
        union{
            base atom;
            ptrs pair;
        }node;
    };

    typedef exp *list;

    list head(const list s);
    list tail(const list s);
    list cons(const list h, const list t);
    list makeAtom(const base x);
    bool isAtom(const list s);
    bool isNull(const list s);
    void destroy(list s);

    void read_list(list& y);
    void read_exp(base prev, list& y);
    void read_seq(list& y);
    void write_list(const list x);
    void write_seq(const list x);
}
```

Название файла: lisp.cpp

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

using namespace std;

#include "list.h"

namespace h_list{
    list head(const list s){
        if(s){
            if(!isAtom(s)){
```

```
            return s->node.pair.head;
        }
        else{
            cerr << "Error: Head(atom)\n";
            exit(1);
        }
    }
    else{
        cerr << "Error: Heid(NULL)\n";
        exit(1);
    }
}

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

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

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

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

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

void destroy(list s){
    if(s){
        if(!isAtom(s)){
            destroy(head(s));
            destroy(tail(s));
        }
        delete s;
    }
}

void read_list(list& y){
    base x;
    do
        cin >> x;
    while(x == ' ');
    read_exp(x, y);
}

void read_exp(base prev, list& y){
    if(prev == ')'){
        cerr << " ! List.Error 1 " << endl;
        exit(1);
    }
    else if(prev != '(')
        y = make_atom(prev);
    else
        read_seq(y);
}

void read_seq(list& y){
    base x;
    list p1, p2;

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

void write_list(const list x){
    if(isNull(x))
        cout << " ()";
    else if(isAtom(x))
        cout << ' ' << x->node.atom;
    else{
        cout << " (";
        write_seq(x);
```

```
            cout << " )";
        }
    }

    void write_seq(const list x){
        if (!isNull(x)){
            write_list(head(x));
            write_seq(tail(x));
        }
    }
}
```