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: « Иерархический список»

Студент гр. 8381 _____ Нгуен Ш. Х.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2019

Цель работы.

Ознакомиться с методами использования рекурсии и реализацией иерархического списка, написать программу преобразования выражения, хранящегося в иерархическом списке, в постфиксную форму с использованием рекурсии, а также обратить иерархический список на всех уровнях вложения.

Основные теоретические положения.

Рекурсия — вызов функции (процедуры) из неё же самой, непосредственно (простая рекурсия) или через другие функции (сложная или косвенная рекурсия), например, функция вызывает функцию , а функция — функцию . Количество вложенных вызовов функции или процедуры называется глубиной рекурсии. Рекурсивная программа позволяет описать повторяющееся или даже потенциально бесконечное вычисление, причём без явных повторений частей программы и использования циклов.

Иерархический список — список имеющий несколько уровней, в нашем случае, понижение уровня происходит при открытии скобки(‘(’), соответственно при закрытии скобки происходит переход на предыдущий уровень.

Рекурсивное определение типа данных S_expr (El) использует данное ранее определение линейного списка (типа L_list):

<S_expr (El)>::=<Atomic (El)> | <L_list (S_expr (El))>

(L_list : линейный список)

< Atomic (El) > ::= < El >

< L_list(El) > ::= < Null_list > | < Non_null_list(El) >

<Null_list> ::= Nil

< Non_null_list(El) > ::= < Pair(El) >

< Pair(El) > ::= (< Head_l (El) > . < Tail_l (El) >)

< Head_l (El) > ::= < El >

< Tail_l (El) > ::= < L_list(El) >

Задание <Вариант 14>

Обратить иерархический список на всех уровнях вложения; например, для исходного списка (a(bc)d) результатом обращения будет список (d(cb)a).

Выполнение работы

В файле lisp.h описаны структуры s_expr и two_ptr, которые вместе описывают структуру элементов списка: первая из них описывает структуру узла списка (union), вторая же хранит пару указателей на начало и конец данного списка. Также с помощью директивы typedef был задан синоним lisp для (s_expr *).

Функции read_lisp(), read_s_expr() и read_seq() для получения входных данных из консоли, настроенных в иерархический список. В функции read_lisp() получает каждый символ x через цикл do-while. Если x = ' ', то команда 'cin >> x; ' выполняется снова, пока x != ' '. Потом функция read_s_expr() вызывается. В функции read_s_expr() если x = ')', то сообщение об ошибке "Error:Missing '(' " и выход. А если x = '(' через функцию cin.peek() мы проверяем, что список или часть списка пустом или нет. Если нет функция read_seq() вызывается. В этой функции мы выполняем рекурсию для построения иерархических списков и объединения «головы» и «хвоста» с помощью функции cons (). Напротив функция make_atom (x) вызывается и элемент возвращаемого списка хранится в переменной y.

Затем были описаны функции работы со списком:

- функция head(const lisp s), возвращающая указатель на начало списка;
- функция tail(const lisp s), возвращающая указатель на конец списка;
- функция isNull(const lisp s), проверяющая, является ли элемент атомом или парой;
- функция isAtom(const lisp s), проверяющая, является ли base
- функция cons(const lisp h, const lisp t), создающая точечную пару

элементов (новый список из «головы» и «хвоста»).

Также были описаны функции работы с элементами списка:

- функция make_atom(const char * x), создающая элемент списка, хранящий переданную в функцию строку;
- функция isAtom(const lisp s), проверяющая, является ли элемент списка обычным узлом, а не началом другого списка.
- функция concat(const lisp y, const lisp z), объединяющая два списка;
- функция reverse(const lisp s), обращающая иерархический список на всех уровнях вложения.

Вывод

В ходе выполнения лабораторной работы была изучена такая структура данных как иерархические списки, а также рекурсивные методы ее обработки. Была реализована программа на C++, использующая иерархические списки.

Тестирование

Входная строка	Строка после форматирования
(ab)	(b a)
(h(e(ll(o)w)o)r(ld))	((d l) r (o (w (o) l l) e) h)
(BA())	Error: List emtry!
abc	Error: Head(atom)
)(ooo)	Error:Missing '('

ПРИЛОЖЕНИЕ А

lisp.h

```
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);

     lisp reverse(const lisp s);
     lisp rev(const lisp s, const lisp z);

}
```

lisp.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);
  }
  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));
          }
     }

     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)));
     }
}
```

main.cpp

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

```
using namespace std;

int main(){
     setlocale (0,"Rus");
     lisp s;
     cout<<boolalpha;
     cout<<"Enter a hierarchical list:  ";
     read_lisp(s);

     lisp z;
     z=reverse(s);
     cout<<"Reverse hierarchical list:  ";
     write_lisp(z);
     cout<<endl;
     return 0;
}
```