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: Рекурсивная обработка иерархических списков

Студентка гр. 8381 _____ Звегинцева Е.Н.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2019

Цель работы.

Познакомиться с основными функциями создания и обработки иерархического списка.

Постановка задачи.

Задание №6.

Проверить иерархический список на наличие в нем заданного элемента (атома) *x*;

Основные теоретические положения.

Представление иерархического списка

Традиционно иерархические списки представляют или графически, или в виде скобочной записи. На рис.1 приведен пример графического изображения иерархического списка. Соответствующая этому изображению сокращенная скобочная запись — это (a (b c) d e).

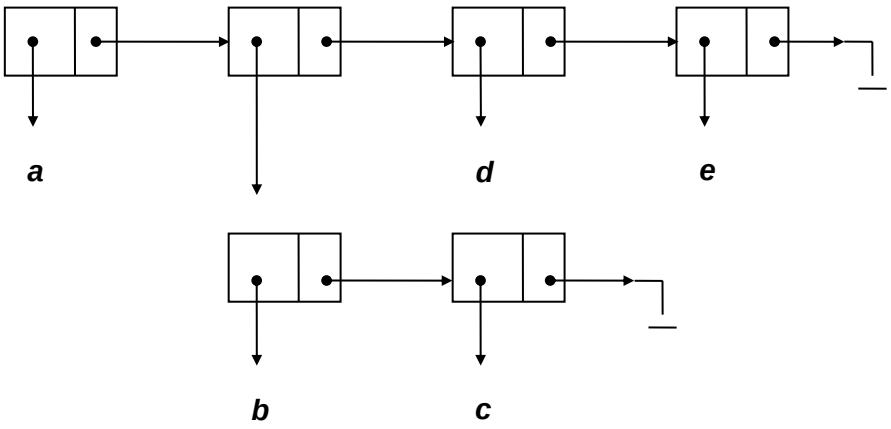

Рисунок 1 - Пример представления иерархического списка в виде двумерного рисунка

Переход от полной скобочной записи, соответствующей определению иерархического списка, к сокращенной производится путем отбрасывания конструкции Nill и удаления необходимое число раз пары скобок вместе с предшествующей открывающей скобке точкой, как показано в табл.1.

Таблица 1 - Примеры перехода от полной к сокращенной скобочной записи иерархических списков.

Полная запись	Сокращенная запись
a	*a*
Nil	( )
(*a* . (*b* . (*c* . *Nil*)))	(*a b c*)
(*a* . ((*b* . (*c* . *Nil*)) . (*d* . (*e* . *Nil*))))	(*a* (*b c*) *d e*)

Согласно приведенному определению иерархического списка, структура непустого иерархического списка — это элемент размеченного объединения множества атомов и множества пар «голова-хвост».

Выполнение работы.

Создаем структуры данных *s_expr*, в которой содержится два поля: *tag* (флаг: 1-атом, 0-узел) и *union node* (хранит в себе значение атома *atom* или структуру *two_pair*), *two_pair*, в которой храниться указатели на атом или узел.

Создаем функции:

- *read_lisp()* - Пропускает пробелы и вызывает функция под номером 2. На вход подается указатель на список и название потока ввода
- *read_s_expr()* - Создает атомы и вызывает функцию с номер 3 для дальнейшей обработки строки. На вход подается последний символ, считанный с потока, указатель на список и название потока ввода.
- *read_seq()* - Рекурсивная функция, которая обрабатывает строку и создает и скрепляет узлы между собой. На вход подается указатель на список и название потока ввода
- *lisp make_atom()* - Создает структуру с атомом. На вход подается имя атом. Возвращает указатель, на созданную струтуру

- *lisp cons()* - Присоединяет узел к списку. На вход подается указатель на голову и хвост узла. Возвращает указатель на присоединенный узел.
- *write_lisp()* - Выводит на экран список в виде атомов и узлов в виде скобок. На вход подается указатель на список и название потока ввода.
- *write_seq()* - Выводит на экран список в виде атомов и узлов в виде скобок.Сама функция выводит непосредственно хвост узла. На вход подается указатель на список и название потока ввода.
- *isAtom()* - Проверяет является ли этот элемент списка атомом или узлом. На вход подается указатель на структуру. Возвращает 1 – если это атом, 0 – если узел.
- *lisp tail()* - Возвращает указатель на tail списка. На вход подается указатель на структуру s.
- *lisp head()* - Возвращает указатель на head списка. На вход подается указатель на структуру s.
- *isNull()* - Проверяет является ли список пустым. На вход подается указатель на структуру s. Возвращает 1 – если пуст, 0 – если не пуст.
- *Check()* - Возвращает 1- если элемент найдем, 0 – если не найден. На вход подается указатель на Иерархический список, Искомый Атом x, начальное значение результата поиска.

Тестовые задания.

Входные данные	Исходные данные
(a (b))	1. Ввод списка вручную 2. Ввод списка из файла 3. Выход Ваш выбор: 1 Введите список: (a(b)) Введенный список: (a (b)) Введите Атом x(искомый атом): b Вызов функции CHECK_ATOM_X __Вызов функции CHECK_HEAD_TO_ATOM_X __Последний считанный символ: a __Проверка на совпадение с символом b __Завершение функции CHECK_HEAD_TO_ATOM_X __Вызов функции CHECK_TAIL_TO_ATOM_X ____Вызов функции CHECK_HEAD_TO_ATOM_X _____Вызов функции CHECK_HEAD_TO_ATOM_X _____Последний считанный символ: b _____Проверка на совпадение с символом b _____Завершение функции CHECK_HEAD_TO_ATOM_X _____Вызов функции CHECK_TAIL_TO_ATOM_X _____Завершение функции CHECK_TAIL_TO_ATOM_X ____Завершение функции CHECK_HEAD_TO_ATOM_X ____Вызов функции CHECK_TAIL_TO_ATOM_X ____Завершение функции CHECK_TAIL_TO_ATOM_X __Завершение функции CHECK_TAIL_TO_ATOM_X Завершение функции CHECK_ATOM_X Атом x присутствует в списке
(x y z)	Введенный список: (x y z) Введите Атом x(искомый атом): a Вызов функции CHECK_ATOM_X __Вызов функции CHECK_HEAD_TO_ATOM_X __Последний считанный символ: x __Проверка на совпадение с символом a __Завершение функции CHECK_HEAD_TO_ATOM_X __Вызов функции CHECK_TAIL_TO_ATOM_X ____Вызов функции CHECK_HEAD_TO_ATOM_X ____Последний считанный символ: y ____Проверка на совпадение с символом a ____Завершение функции CHECK_HEAD_TO_ATOM_X ____Вызов функции CHECK_TAIL_TO_ATOM_X _____Вызов функции CHECK_HEAD_TO_ATOM_X _____Последний считанный символ: z _____Проверка на совпадение с символом a _____Завершение функции CHECK_HEAD_TO_ATOM_X _____Вызов функции CHECK_TAIL_TO_ATOM_X _____Завершение функции CHECK_TAIL_TO_ATOM_X ____Завершение функции CHECK_TAIL_TO_ATOM_X __Завершение функции CHECK_TAIL_TO_ATOM_X Завершение функции CHECK_ATOM_X Атом x не присутствует в списке

(1 2 3 (4))	Введенный список: (1 2 3 (4)) Введите Атом x(искомый атом): 5 Вызов функции CHECK_ATOM_X __Вызов функции CHECK_HEAD_TO_ATOM_X __Последний считанный символ: 1 __Проверка на совпадение с символом 5 __Завершение функции CHECK_HEAD_TO_ATOM_X __Вызов функции CHECK_TAIL_TO_ATOM_X ____Вызов функции CHECK_HEAD_TO_ATOM_X ____Последний считанный символ: 2 ____Проверка на совпадение с символом 5 ____Завершение функции CHECK_HEAD_TO_ATOM_X ____Вызов функции CHECK_TAIL_TO_ATOM_X ______Вызов функции CHECK_HEAD_TO_ATOM_X ______Последний считанный символ: 3 ______Проверка на совпадение с символом 5 ______Завершение функции CHECK_HEAD_TO_ATOM_X ______Вызов функции CHECK_TAIL_TO_ATOM_X ________Вызов функции CHECK_HEAD_TO_ATOM_X __________Вызов функции CHECK_HEAD_TO_ATOM_X __________Последний считанный символ: 4 __________Проверка на совпадение с символом 5 __________Завершение функции CHECK_HEAD_TO_ATOM_X __________Вызов функции CHECK_TAIL_TO_ATOM_X __________Завершение функции CHECK_TAIL_TO_ATOM_X ________Завершение функции CHECK_HEAD_TO_ATOM_X ________Вызов функции CHECK_TAIL_TO_ATOM_X ________Завершение функции CHECK_TAIL_TO_ATOM_X ______Завершение функции CHECK_TAIL_TO_ATOM_X ____Завершение функции CHECK_TAIL_TO_ATOM_X __Завершение функции CHECK_TAIL_TO_ATOM_X Завершение функции CHECK_ATOM_X Атом x не присутствует в списке
(d f g (f g h))	Введите список: (d f g (fgh)) Введенный список: (d f g (f g h)) Введите Атом x(искомый атом): k Вызов функции CHECK_ATOM_X __Вызов функции CHECK_HEAD_TO_ATOM_X __Последний считанный символ: d __Проверка на совпадение с символом k __Завершение функции CHECK_HEAD_TO_ATOM_X __Вызов функции CHECK_TAIL_TO_ATOM_X ____Вызов функции CHECK_HEAD_TO_ATOM_X ____Последний считанный символ: f ____Проверка на совпадение с символом k ____Завершение функции CHECK_HEAD_TO_ATOM_X ____Вызов функции CHECK_TAIL_TO_ATOM_X ______Вызов функции CHECK_HEAD_TO_ATOM_X ______Последний считанный символ: g ______Проверка на совпадение с символом k ______Завершение функции CHECK_HEAD_TO_ATOM_X ______Вызов функции CHECK_TAIL_TO_ATOM_X ________Вызов функции CHECK_HEAD_TO_ATOM_X __________Вызов функции CHECK_HEAD_TO_ATOM_X __________Последний считанный символ: f __________Проверка на совпадение с символом k __________Завершение функции CHECK_HEAD_TO_ATOM_X __________Вызов функции CHECK_TAIL_TO_ATOM_X ____________Вызов функции CHECK_HEAD_TO_ATOM_X ____________Последний считанный символ: g ____________Проверка на совпадение с символом k ____________Завершение функции CHECK_HEAD_TO_ATOM_X ____________Вызов функции CHECK_TAIL_TO_ATOM_X ______________Вызов функции CHECK_HEAD_TO_ATOM_X ______________Последний считанный символ: h ______________Проверка на совпадение с символом k ______________Завершение функции CHECK_HEAD_TO_ATOM_X ______________Вызов функции CHECK_TAIL_TO_ATOM_X ______________Завершение функции CHECK_TAIL_TO_ATOM_X ____________Завершение функции CHECK_TAIL_TO_ATOM_X __________Завершение функции CHECK_TAIL_TO_ATOM_X ________Завершение функции CHECK_HEAD_TO_ATOM_X ________Вызов функции CHECK_TAIL_TO_ATOM_X ________Завершение функции CHECK_TAIL_TO_ATOM_X ______Завершение функции CHECK_TAIL_TO_ATOM_X ____Завершение функции CHECK_TAIL_TO_ATOM_X __Завершение функции CHECK_TAIL_TO_ATOM_X Завершение функции CHECK_ATOM_X Атом x не присутствует в списке

Проверка на пустой файл	1. Ввод списка вручную 2. Ввод списка из файла 3. Выход Ваш выбор: 2 Файл не открыт

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <iostream>
#include <cstdlib>
#include <fstream>
using namespace std;
struct s_expr;
int c=0;
struct  two_ptr
{
      s_expr *hd;
      s_expr *tl;
} ;   //end two_ptr;
struct s_expr
{
      bool tag; // true: atom, false: pair
      union
      {
            char atom;
            two_ptr pair;
      } node;          //end union node
};               //end s_expr
typedef s_expr *lisp;
// функции
void print_s_expr( lisp s );
// базовые функции:
lisp head (const lisp s);
lisp tail (const lisp s);
lisp cons (const lisp h, const lisp t);
lisp make_atom (const char x);
bool isAtom (const lisp s);
bool isNull (const lisp s);
void destroy (lisp s);

// функции ввода:
void read_lisp ( lisp& y,istream & in=cin);           // основная
void read_s_expr (char prev, lisp& y,istream & in);
void read_seq ( lisp& y,istream & in);

// функции вывода:
void write_lisp (const lisp x);       // основная
void write_seq (const lisp x);

void Space(int c) // создание отступов для вызовов/завершений фукнций
{
      for (int j=1; j <=c; j++) cout <<"__";
}

lisp head(const lisp s)
```

```
{// PreCondition: not null (s)
      if (s != NULL)
      {
            if (!isAtom(s))
            {
                  return s->node.pair.hd;
            }
            else
            {
                  cout << "Error: Head(atom) \n"; exit(1);
            }
      }
      else
      {
            cout << "Error: Head(nil) \n";
            return 0;
      }
}

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
            {
                  cout << "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: Tail(nil) \n";
		exit(1);
	}
	else
	{
		p = new s_expr;
		//cout << "node:  " << p << endl << endl;
		if ( p == NULL)
		{
			cerr << "Error: 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 char x)
	{
		//cout << "Current simbol: " << x << endl;
		lisp s;
		s = new s_expr;
		s -> tag = true;
		s->node.atom = x;
		//cout << "Addres simbol: "  <<  s <<  endl;
		return s;
	}



// ввод списка с консоли
void read_lisp ( lisp & y,istream & in)
	{
		char x;
		do
```

```
            {
                  in >> x;
            }while (x==' ');

            read_s_expr ( x, y,in);
      } //end read_lisp

void read_s_expr (char prev, lisp & y,std::istream & in)
      { //prev - ранее прочитанный символ
            if ( prev == ')' )
            {
                  cerr << " ! List.Error 1 " << endl;
                  exit(1);
            }
            else
            if ( prev != '(' )
            {
                  y = make_atom (prev);
            }
            else
            {
                  read_seq (y,in);
            }
      } //end read_s_expr

void read_seq ( lisp& y,std::istream & in)
      {
            char x;
            lisp p1, p2;
            if (!(in >> x)) x=0;
            if((in.eof()||(x==0)))
            {
                  cerr << " ! List.Error 2 " << endl;
                  exit(1);
            }else
            {
                  //cout << "Current simbol:" << x << endl;
                  while  (x == ' ')
                  {
                        in >> x;
                  }

                  if ( x == ')' )
                  {
                        y = NULL;
                  }else
                  {
                        read_s_expr ( x, p1,in);
                        read_seq ( p2,in);
                        y = cons (p1, p2);
                  }
```

```
            }
      } //end read_seq

// Процедура вывода списка с обрамляющими его скобками - write_lisp,
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));
      }
}
bool Check(const lisp HierarchList,char x,bool b)
{
      if(isNull(HierarchList)||(b)) // Если список закончился или наше
условие выполнено
      {
            return b;
      }
      else
      {
            if(HierarchList->tag) // Если текущий символ - атом
            {
                  Space(c);
                  cout << "Последний считанный символ: " <<
HierarchList->node.atom << endl;
                  Space(c);
                  cout << "Проверка на совпадение с символом " << x <<
endl;
                  if (HierarchList->node.atom==x)//При совпадение с
нашим элементом
                  {
                        b=true;
```

```
                    return b;
                }
            }
            else
            {
                c++;
                Space(c); // Создание отступов для наглядности глубины
вызова функции
                cout << "Вызов функции CHECK_HEAD_TO_ATOM_X" << endl;
                b=Check(HierarchList->node.pair.hd,x,b);
                Space(c); // Создание отступов для наглядности глубины
вызова функции
                cout << "Завершение функции CHECK_HEAD_TO_ATOM_X" <<
endl;
                c--;
                c++;
                Space(c); // Создание отступов для наглядности глубины
вызова функции
                cout << "Вызов функции CHECK_TAIL_TO_ATOM_X" << endl;
                b=Check(HierarchList->node.pair.tl,x,b);
                Space(c); // Создание отступов для наглядности глубины
вызова функции
                c--;
                cout << "Завершение функции CHECK_TAIL_TO_ATOM_X" <<
endl;
            }
            return b;
      }
}
int main(void)
{
      setlocale(0, "");
      int i=0;
      lisp HierarchList=NULL;
      ifstream fin;
      bool k=true;
      while(k)
      {
            cout << endl <<
"_____________________________________________________" << endl;
            cout << "1. Ввод списка вручную\n"    << "2. Ввод списка из
файла\n"<< "3. Выход\n" << "Ваш выбор: ";
            cin >> i;
            while(cin.fail())// Проверка на ошибки ввода
            {
                cin.clear();
                cin.sync();
                cout << "Ошибка номера!\n";
                cout << "Ваш выбор: ";
                cin >> i;
            }
```

```
            switch(i)
            {
                  case 1:
                        std::cout << "Введите список:" << std::endl;
                        read_lisp(HierarchList);//Считываем с консоли,
поток по умолчанию cin
                        break;
                  case 2:
                        fin.open("atom.txt");
                        if(!fin.is_open())
                        {
                              cout << "Файл не открыт\n";
                              system("pause");
                              return 1;
                        }
                        read_lisp (HierarchList,fin);//Считываем из файла
указывая поток fin
                        fin.close();
                        break;
                  case 3:
                        k=false;
                        break;
            }
            if (!k) break;
            cout << "Введенный список: "  ;
            write_lisp (HierarchList);// Выводим список на экран
            char x;
            cout << endl << "Введите Атом x(искомый атом): ";
            cin >> x;
            cout << endl;
            bool b=false;
            Space(c); // Создание отступов для наглядности глубины
вызова функции
            cout << "Вызов функции CHECK_ATOM_X" << endl;
            b=Check(HierarchList,x,b);
            Space(c); // Создание отступов для наглядности глубины
вызова функции
            cout << "Завершение функции CHECK_ATOM_X" << endl;
            if(b)
            {
                  cout << endl << "Атом x присутствует в списке";
            }
            else
            {
                  cout << endl << "Атом x не присутствует в списке";
            }
      }
      cin.get();
      cin.get();
      return 0;
}
```