**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**

|  |  |  |
| --- | --- | --- |
| Студент гр. 6383 |  | Михеева Е.Е. |
| Преподаватель |  | Шолохова О. М. |

Санкт-Петербург

2017

**Цель работы.**

Познакомиться с одной из часто используемых на практике нелинейных конструкций, способами её организации и рекурсивной обработки. получить навыки решения задач обработки иерархических списков, как с использованием базовых функций их рекурсивной обработки, так и без использования рекурсии.

**Постановка задачи.**

Задание предполагает самостоятельную разработку студентом одного или нескольких модулей на языке С++, реализующих согласованный с преподавателем набор операций над иерархическими списками, а также главной программы, непосредственно решающей поставленную задачу. Предполагается выполнение задания в двух вариантах: с использованием базовых функций рекурсивной обработки иерархических списков и без использования рекурсии.

Во всех случаях, когда это не оговорено особо, предполагается, что исходные и результирующие списки размещаются в файлах подходящего типа. Для представления иерархических списков рекомендуется использовать сокращенную скобочную запись.

Задание: 3. заменить в иерархическом списке все вхождения заданного элемента (атома) x на заданный элемент (атом) y;

**Основные теоретические положения.**

Традиционно иерархические списки представляют или графически или в виде скобочной записи. На рисунке 2.1 приведен пример графического изображения иерархического списка. Соответствующая этому изображению сокращенная скобочная запись － это (a (b c) d e).

Рис. 2.1. Пример представления иерархического списка в виде двумерного рисунка.

Определим соответствующий тип данных S\_expr (El) рекурсивно, используя определение линейного списка (типа L\_list):

< S\_expr (El) > ::= < Atomic (El) > | < L\_list (S\_expr (El)) >,

< Atomic (E) > ::= < El >.

< *L\_list*(*El*) > ::= < *Null\_list* > | < *Non\_null\_list*(*El*) >

< *Null\_lis t*> ::= *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*) >

Функциональная спецификация иерархического списка включает:

функции － селекторы Head и Tail , выделяющие соответственно «голову» и «хвост» списка

функции － конструкторы: Cons, создающая точечную пару (новый список из «головы» и «хвоста»), и Make\_Atom, создающая атомарное S-выражение.

предикаты Is Null, проверяющий список на отсутствие в нем элементов, и Atom, проверяющий, является ли список атомом.

Функция Destroy, позволяющая уничтожить созданный список, т.е. освободить память от ставших ненужными списочных структур.

**Спецификация программы.**

*Назначение программы*.

Программа предназначается для подсчета количества в иерархическом списке различных атомов.

*Описание программы*.

Программа написана на языке C++ с использованием компилятора gcc. Входными данными для программы является: строка, представляющая список, элемент для замены (символ), заданный элемент (символ). Входные данные вводятся из консоли.

Выходными данными являются:

В случае соответствия исходной строки сокращенной скобочной записи иерархического списка — сокращенная скобочная запись результирующего списка в виде строки, записанной в файле output.txt, расположенном в каталоге с программой. В случае несоответствия — сообщение с кодом ошибки, выведенное на консоль.

*Пример исходных данных*

*(a b b c)*

*Пример выходных данных*

2

(a c)

*Реализация*.

void find1(lisp s, base ch, bool found);

void find1\_seq(lisp s, base ch, bool found);

Рекурсивные функции для поиска повторяющихся элементов

Предусловие: s — содержит входной список.

Итеративная функция для замены всех атомов в списке равных from, на атомы to, в случае если максимальная глубина вложенности превышает константу STACK\_SIZE, выводится сообщение stack overflow.

Предусловие: list — содержит входной список.

Постусловие: list — содержит выходной список.

**Тестирование.**

Во всех тестах в таблице предполагается, что from=“а“, to=“X“.

|  |  |
| --- | --- |
| Входная строка | Ожидаемый результат |
| (a) | 1  (a) |
| (b(aa)) | 1  (b) |
| (a a c c b(dk)) | 3  ( b d k ) |

**Выводы.**

В ходе работы я познакомилась с одной из часто используемых на практике нелинейных конструкций, способами её организации и рекурсивной обработки, получила навыки решения задач обработки иерархических списков, как с использованием базовых функций их рекурсивной обработки, так и без использования рекурсии.

Приложение A. Код программы.

Файл main.cpp

// ConsoleApplication36.cpp: определяет точку входа для консольного приложения.

//

#include "stdafx.h"

#include <iostream>

#include <cstdlib>

#include "l\_intrfc.h"

//#include <windows.h>

using namespace std;

using namespace h\_list;

void find1\_seq(lisp x, base ch, bool &found,int &r);

void find1(lisp x, base ch, bool &found,int &r);

void find1(lisp x, base ch, bool &found, int &r)

{

if (isNull(x)) return;

else if (isAtom(x)) {

if (x->node.atom == ch)

{

r++;

if (r > 1) found = true;

};// return;

}

else {

find1\_seq( x, ch, found,r);

// return ;

}

}

//...........................

void find1\_seq(lisp x, base ch, bool &found, int &r)

{

if (!isNull(x)) {

find1(head(x), ch, found,r);

find1\_seq(tail(x), ch, found,r);

}

}

//......................

struct n {

base ch;

n \*tail;

//n \*head;

};

typedef n \*p;

void f\_seq(lisp x, lisp &y, p head);

lisp f(lisp x, lisp &y, p head);

p find(p head, base c, bool &found);

lisp f(lisp x, lisp &y,p head)

{

if (isNull(x)) return nullptr;

else if (isAtom(x)) {

bool found = false;

find(head, x->node.atom, found);

if (found == true)

{

y = cons(make\_atom(x->node.atom),y);

};// return;

}

else {

f\_seq(x, y,head);

return y;

}

}

void f\_seq(lisp x, lisp &y,p head1)

{

if (!isNull(x)) {

f(head(x), y,head1);

f\_seq(tail(x), y,head1);

}

}

p tail\_l(p head)

{

return head->tail;

}

p write\_l(p head)

{

if (head != NULL)

{

//if (head == nullptr) return nullptr;;

cout << head->ch << endl;

write\_l(tail\_l(head));

}

else return nullptr;

}

p find(p head, base c, bool &found) // found true - найден

{

if (head != nullptr) {

if (head->ch == c) {

found = true;

return NULL;

}

else find(tail\_l(head), c, found);

}

else return NULL;

}

p change( lisp s,lisp s1,p &t, p &head,int &k);

void change\_seq( lisp s,lisp s1,p &t,p &head,int &k);

//p s4;

//lisp s1;

int main()

{

setlocale(0, "rus");

lisp s1,s2, s3;

p s4;

// cout << boolalpha;

p head;

int k = 0;

cout << "введите list1:" << endl;

read\_lisp(s1);

cout << "введен list1: " << endl;

write\_lisp(s1);

cout << endl;

cout << "change:\n";

change(s1,s1, s4, head,k);

//write\_lisp();

cout << k << endl;

if (k == 0) {

cout << "()"; system("pause"); return 0;

};

s4->tail = nullptr;

//write\_l(head);

cout << "flatten списка = " << endl;

s3 = nullptr;

s3 = f(s1, s3,head);

// s3 = flatten1(s1,head);

write\_lisp(s3);

cout << endl;

system("pause");

return 0;

}

//

p change( lisp x, lisp s1,p &t, p &head,int &k)

p u;

if (isNull(x)) cout << " ()";

else if (isAtom(x)) {

int r1 = 0;

bool found = false;

find1(s1, x->node.atom, found,r1);

if (found == false) {

k++;

u = new (n);

if (k == 1) head = u;

if (k > 1) t->tail = u;

t = u;

t->tail = nullptr;

t->ch = x->node.atom;

}

return nullptr;

}

else {

change\_seq(x,s1,t,head,k);

return t;

}

}

void change\_seq( lisp x, lisp s1,p &t, p &head1,int &k)

{

if (!isNull(x)) {

change(head(x),s1,t,head1,k);

change\_seq(tail(x),s1,t,head1,k);

}

}

Файл #include "l\_intrfc.h"

#ifndef L\_INTRFC\_H

#define L\_INTRFC\_H

namespace h\_list

{

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 );

// 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); // основная

void read\_s\_expr (base prev, lisp& y);

void read\_seq ( lisp& y);

// функции вывода:

void write\_lisp (const lisp x); // основная

void write\_seq (const lisp x);

lisp copy\_lisp (const lisp x);

} // end of namespace h\_list

// continue of namespace h\_list

#include "l\_intrfc.h"

#include <iostream>

#include <cstdlib>

using namespace std;

namespace h\_list

{

//....................................

lisp head (const lisp s)

{// PreCondition: not null (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 == nullptr) 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 { 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 == 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;

}

//...........................

void destroy (lisp s)

{

if ( s != NULL) {

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)

{ base x;

do cin >> x; while (x==' ');

read\_s\_expr ( x, y);

} //end read\_lisp

//...........................

void read\_s\_expr (base prev, lisp& y)

{ //prev － ранее прочитанный символ}

if ( prev == ')' ) {cerr << " ! List.Error 1 " << endl; exit(1); }

else if ( prev != '(' ) y = make\_atom (prev);

else read\_seq (y);

} //end read\_s\_expr

//...........................

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);

}

}

} //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 << endl;

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));

}

}

//...........................