**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МО ЭВМ**

**ОТЧЕТ**

**по лабораторной работе № 2**

**по дисциплине «Программирование»**

**Тема:**

**Рекурсивная обработка иерархических списков**

|  |  |  |
| --- | --- | --- |
| Студент гр. 6383 |  | Быков И. В. |
| Преподаватель |  | Шолохова О.М. |

Санкт-Петербург

2017

1. **Цель работы.**

Ознакомиться 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.

1. **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.

Вариант 3.

Заменить в иерархическом списке все вхождения заданного элемента (атома) x на заданный элемент (атом) y;

1. **Основные теоретические положения.**

В практических приложениях возникает необходимость работы с более сложными, чем линейные списки, нелинейными конструкциями. Рассмотрим одну из них, называемую иерархическим списком элементов базового типа El или S-выражением.

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

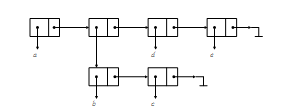
< Non\_null\_list(El) > ::= < Pair(El) >

< Pair(El) > ::= ( < Head\_l(El) > . < Tail\_l(El) > )

< Head\_l(El) > ::= < El >

< Tail\_l(El) > ::= < L\_list(El) >

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

1. **Спецификация программы.**

Программа предназначена для замены всех вхождений заданного атома на заданный атом.

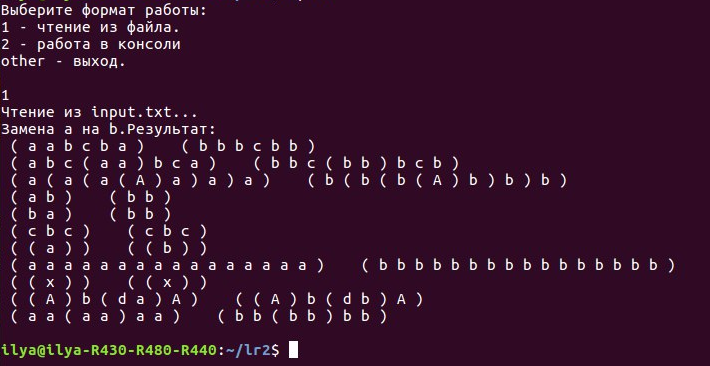
Программа написана на языке C++ с использованием компилятора mingw32- g++. Входными данными для программы являются:

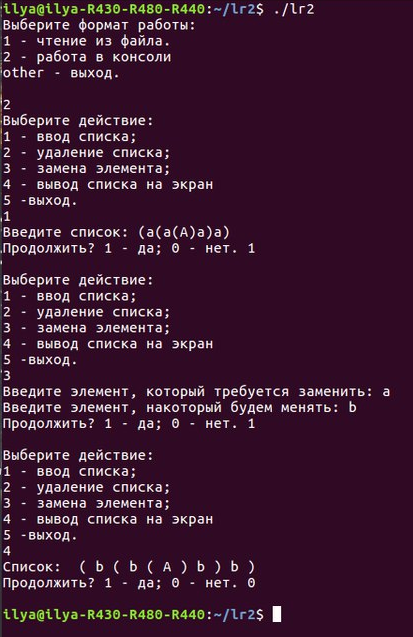
элемент типа char – атом, который требуется заменить;

элемент типа char – атом, на который будем менять;

иерархический список в сокращенной скобочной записи.

1. **Пример диалога с пользователем.**

****

****

1. **Реализация.**

Функции:

void myReplace(lisp list, const base &a, const base &b);

*Функция, которая заменяет все вхождения атома а на атом b*

*предусловие: a – заменяемый атом*

*b – атом, на который меняем*

*list – иерархический список в сокращенной скобочной записи*

1. **Тестирование.**

Делаем замену а на b

|  |  |
| --- | --- |
| (a a b c b a) | (b b b c b b) |
| (a b c (a a) b c a) | (b b c ( b b ) b c b) |
| (a(a(a(A)a)a)a) | (b(b(b(A)b)b)b) |
| (a b) | (b b) |
| (b a) | (b b) |
| (c b c) | (c b c) |
| (a) | (b) |
| (a a a a a a a a) | (b b b b b b b b) |
| (x) | (x) |
| ((A) b (d a) A) | ((A) b (d b) A) |
| (a a( a a) a a) | (b b (b b) b b) |

**Выводы****.**

В результате выполнения лабораторной работы мною были получены основные сведения о иерархических списках. Ознакомился с базовыми функциями обработки иерархических списков. Была написана функция, позволяющая заменить все вхождения заданного атома на заданный. Я выбрал рекурсивный способ написания функции, так как он наиболее прост и понятен в реализации.

**Приложение А. Исходный код.**

**А.1. list.h**

#ifndef LIST\_H\_INCLUDED

#define LIST\_H\_INCLUDED

#include <fstream>

using namespace std;

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, istream& in); // основная

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

void read\_seq ( lisp& y, istream& in);

// функции вывода:

void write\_lisp (const lisp x, ostream& out); // основная

void write\_seq (const lisp x, ostream& out);

lisp copy\_lisp (const lisp x);

} // end of namespace h\_list

#endif // LIST\_H\_INCLUDED

A.2. list.cpp

#include "list.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 == 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 { 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, istream& in)

{ base x;

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

read\_s\_expr ( x, y, in);

} //end read\_lisp

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

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

{ base x;

lisp p1, p2;

if (!(in >> x)) {cerr << " ! List.Error 2 " << endl; exit(1);}

else {

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,

// а без обрамляющих скобок － write\_seq

void write\_lisp (const lisp x, ostream& out)

{//пустой список выводится как ()

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

else if (isAtom(x)) out << ' ' << x->node.atom;

else { //непустой список}

out << " (" ;

write\_seq(x, out);

out << " )";

}

} // end write\_lisp

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

void write\_seq (const lisp x, ostream& out)

{//выводит последовательность элементов списка без обрамляющих его скобок

if (!isNull(x)) {

write\_lisp(head (x), out);

write\_seq(tail (x), out);

}

}

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

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)));

} //end copy-lisp

} // end of namespace h\_list

A.3. main.cpp

/\*Быков И. В. 6383 лабораторная работа #2. вариант 3.

Заменить в иерархическом списке все вхождения заданного элемента (атома) x на заданный элемент (атом) y;

использование модуля с АТД "Иерархический Список" .

Интерфейс модуля в заголовочном файле "list.h"

и его реализация (в отдельном файле list.cpp) образуют

пространство имен namespace h\_list

\*/

#include <iostream>

#include <cstdlib>

#include "list.h"

#include <windows.h>

#include <clocale>

#include <io.h>

#include <fcntl.h>

using namespace std;

using namespace h\_list;

/\*

Рекурсивная функция замены всех вхождений заданного атома на заданный

предусловие: список, элемент, который меняем, элемент на который меняем

\*/

void myReplace(lisp list, const base &a, const base &b);

int readAction();

int main ( )

{

//setlocale(LC\_ALL, "Russian");

SetConsoleCP(1251);// установка кодовой страницы win-cp 1251 в поток ввода

SetConsoleOutputCP(1251); // установка кодовой страницы win-cp 1251 в поток вывода

ifstream in( "input.txt");

ofstream out("output.txt");

char a, b; // a - что меняем, b - на что меняем

lisp s;

int flag, format;

cout << "Выберите формат работы:\n1 - чтение из файла.\n2 - работа в консоли\nother - выход.\n" << endl;

cin >> format;

switch (format)

{

case 1:

if (!in) cerr << "Не найден файл input.txt\n";

cout << "Чтение из input.txt..." << endl;

in >> a;

in >> b;

cout << "Замена " << a << " на " << b <<".Результат: " << endl;

while (in.get() != EOF)

{

read\_lisp (s, in);

myReplace(s, a, b);

write\_lisp (s, out);

write\_lisp (s, cout);

out << endl;

cout << endl;

//cout << "destroy list: " << endl;

destroy (s);

//cout << "end! " << endl;

}

break;

case 2:

while (true)

{

switch (readAction())

{

case 1:

cout << "Введите список: ";

read\_lisp (s, cin);

break;

case 2:

destroy(s);

cout << "Cписок удален!" << endl;

break;

case 3:

cout << "Введите элемент, который требуется заменить: ";

cin >> a;

cout << "Введите элемент, накоторый будем менять: ";

cin >> b;

myReplace(s, a, b);

break;

case 4:

cout << "Cписок: ";

write\_lisp (s, cout);

break;

case 5:

return 0;

}

cout << "Продолжить? 1 - да; 0 - нет. ";

cin >> flag;

cout << endl;

if (flag == 0)

return 0;

}

break;

}

}

void myReplace(lisp list, const base &a, const base &b)

{

if (isNull(list))

return;

else if (isAtom(list))

{

if (list->node.atom == a)

list->node.atom = b;

}

else

{

myReplace(head(list), a, b);

myReplace(tail(list), a, b);

}

return;

}

int readAction()

{

while (true)

{

cout <<"Выберите действие:\n1 - ввод списка;\n2 - удаление списка;\n3 - замена элемента;\n4 - вывод списка на экран\n5 -выход.\n";

int action;

cin >> action;

if (action > 5 || action < 1)

{

printf("Ощибка!\n");

continue;

}

return action;

}

}