**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра математического обеспечения и применения ЭВМ**

отчет

**по лабораторной работе №2**

**по дисциплине «Алгоритмы и структуры данных»**

**Тема: Иерархические списки**

|  |  |  |
| --- | --- | --- |
| Студент гр. 7383 |  | Кирсанов А.Я. |
| Преподаватель |  | Размочева Н.В. |

Санкт-Петербург

2018

**Содержание**

[Цель работы 3](#__RefHeading__2318_1303384392)

[Реализация задачи](#__RefHeading__2320_1303384392) 3

[Тестирование](#__RefHeading__2322_1303384392) 4

[Вывод](#__RefHeading__2324_1303384392) 4

[Приложение A. Тестовые случаи](#__RefHeading__4504_1303384392) 5

[Приложение Б. Исходный код программы](#__RefHeading__4506_1303384392) 7

Цель работы

Цель работы: научиться реализовывать иерархический список, освоить базовые функции работы с ним, научиться писать свои функции на основе рекурсии для работы с иерархическим списком.

Формулировка задачи: заменить в иерархическом списке все вхождения заданного элемента (атома) x на заданный элемент (атом) y.

Реализация задачи

Программа состоит из функции main и функций работы с иерархическим списком.

Список состоит из структур. Каждая структура может содержать либо атом (в нашем случае атом – это символ), либо указатель на следующую и предыдущую структуры.

В функции main реализованы два способа ввода списка: с клавиатуры и из файла. Введенная или считанная из файла строка списка передается в поток strstream, затем вызывается функция read\_lisp. Она пропускает пробелы и вызывает функцию read\_s\_expr, передавая в нее первый символ, не являющийся пробелом. read\_s\_expr создает в списке атом если переданный символ не ‘(’ или вызывает функцию read\_seq, которая создает две структуры списка. В каждую из них, посредством вызова функций read\_seq, read\_s\_expr и cons может быть записан либо атом, либо указатель NULL, либо указатели на следующие две структуры. Процесс записи происходит до окончания списка, затем в main вызывается функция печати списка write\_lisp. Эта функция печатает атом, если структура списка является им, либо вызывает функцию write\_seq печати содержимого внутри скобок.

Функция rep принимает на вход список и два атома, введенные на предыдущем шаге. Если список пуст, функция возвращает false. Если спискок указывает на атом, функция сравнивает его с заменяемым и заменяет его на атом y, если сравнение истинно. Иначе возвращает false. Если в голове нет указателя на атом, функция возвращает логическую сумму функций rep, где в качестве первого аргумента указатели на хвост или на голову. Если найдется искомый атом, он будет заменен и функция вернет в main true. В таком случае main выведет "Replaced successfully" и выведет измененный список. Иначе main выведет "No replacement items found".

Тестирование

Сборка и тестирование программы производилось в среде разработки QT на Linux Ubuntu 16.04 LTS.

В ходе тестирование были использованы различные выражения, заведомо правильные или неправильные. Результаты тестирования представлены в приложении А.

При тестировании программы было обнаружено, что функция rep не всегда находит все заменяемые атомы. Ошибка возникла из-за того, что при рекурсивном вызове передавался только указатель head и некоторые атомы не были проверены из-за этого. Решением было возвращать логическую сумму функций rep с указателями на head и на tail элемента.

Вывод

В ходе работы были изучены иерархические списки. Получен опыт работы с базовыми функциями таких списков. Реализована рекурсивная функция поиска заданного атома для последующей замены его определенным значением. Найдены и исправлены ошибки в работе функции.

ПРИЛОЖЕНИЕ А.

ТЕСТОВЫЕ СЛУЧАИ

Таблица 1 — Тестовые случаи

|  |  |
| --- | --- |
| Входные данные | Вывод программы |
| 1 - Reading from file, 2 - Keyboard input, 3 - Exit from the program. 2 Enter a list: List entered: () Enter x y a d | No replacement items found. |
| 1 - Reading from file, 2 - Keyboard input, 3 - Exit from the program. 2 Enter a list: (dsa) List entered: ( d s a ) Enter x y d i | Replaced successfully. Processed list: ( i s a ) |
| 1 - Reading from file, 2 - Keyboard input, 3 - Exit from the program.  1  Enter file name:  input  Input file not open!  ()  Enter x y  a d | No replacement items found. |
| 1 - Reading from file, 2 - Keyboard input, 3 - Exit from the program.  2  Enter a list:  (s(ks)sqew(as(sdd)sd)xzxz)  List entered:  ( s ( k s ) s q e w ( a s ( s d d ) s d ) x z x z )  Enter x y  z u | Replaced successfully.  Processed list:  ( s ( k s ) s q e w ( a s ( s d d ) s d ) x u x u ) |

Окончание таблицы

|  |  |
| --- | --- |
| 1 - Reading from file, 2 - Keyboard input, 3 - Exit from the program.  1  Enter file name:  input  ( a s d f g h j g f d s )  Enter x y  g t | Replaced successfully.  Processed list:  ( a s d f t h j t f d s ) |

**ПРИЛОЖЕНИЕ Б.**

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

**main.cpp:**

#include "l\_intrfc.h"

#include <fstream>

#include <iostream>

#include <cstdlib>

#include <cstdlib>

#include <strstream>

#define N 1000

using namespace h\_list;

using namespace std;

int main()

{

base x, y;

lisp s = NULL;

int k = 0;

char str[N];

while(k != 3){

strstream st;

cout << endl << "1 - Reading from file, 2 - Keyboard input, 3 - Exit from the program." << endl;

cin >> k;

switch (k) {

case 1:{

cout << "Enter file name:" << endl;

cin >> str;

ifstream outfile(str);

if (!outfile) { cout << "Input file not open!" << endl; break; }

outfile.read(str, N);

outfile.close();

st << str;

break;

}

case 2:{

cout << "Enter a list:" << endl;

cin.get();

cin.getline(str, N);

st << str;

break;

}

case 3:{ cout << "Press Enter\n"; return 0; }

}

try { read\_lisp(s, st); }

catch(int a){

Error(a);

continue;

}

cout << "List entered: " << endl;

write\_lisp (s);

cout << endl;

cout << "Enter x y\n";

cin >> x >> y;

if(rep(s, x, y)){

cout << "Replaced successfully." << endl;

cout << "Processed list:" << endl;

write\_lisp (s);

}

else cout << "No replacement items found." << endl;

cout << endl;

destroy(s);

s = NULL;

}

return 0;

}

**l\_impl.cpp:**

#include <iostream>

#include <strstream>

#include "l\_intrfc.h"

using namespace std;

namespace h\_list

{

bool rep(lisp s, base x, base y){

if(isNull(s)) return false;

if(isAtom(s)){

if(getAtom(s) == x){ s->node.atom = y; return true; }

return false;

}

else return rep(head(s), x, y) + rep(tail(s), x, y);

}

lisp head (const lisp s)

{

return s->node.pair.hd;

}

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)

{

return s->node.pair.tl;

}

base getAtom (const lisp s)

{

return s->node.atom;

}

lisp cons (const lisp h, const lisp t)

{lisp p;

p = new s\_expr;

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;

};

}

void read\_lisp ( lisp& y, strstream &st)

{ base x = 0;

if(!(st >> x)) throw 3;

while (x==' '){ st >> x; }

if(x)

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

}

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

{

if(prev == ')') throw 1;

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

else read\_seq (y, st);

}

void read\_seq ( lisp& y, strstream &st)

{ base x;

lisp p1, p2;

if(!(st >> x)) throw 2;

while ( x==' ' ){ st >> x; }

if ( x == ')' ) y = NULL;

else {

read\_s\_expr ( x, p1, st);

read\_seq ( p2, st);

y = cons (p1, p2);

}

}

void write\_lisp (const lisp x)

{

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));

}

}

void Error(int a){

switch (a) {

case 1:{ cout << "no expression before ')'" << endl; break; }

case 2:{ cout << "no ')'" << endl; break; }

case 3:{ cout << "empty expression" << endl; break; }

}

}

}

**l\_intrfc.h:**

// интерфейс АТД "Иерархический Список"

#include <strstream>

#define N 1000

using namespace std;

namespace h\_list

{

typedef char base;

struct s\_expr;

struct two\_ptr

{

s\_expr \*hd;

s\_expr \*tl;

};

struct s\_expr {

bool tag; // true: atom, false: pair

union{

base atom;

two\_ptr pair;

} node;

};

typedef s\_expr \*lisp;

bool rep(lisp s, base x, base y);

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);

void read\_lisp ( lisp& y, strstream &st);

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

void read\_seq ( lisp& y, strstream &st);

void write\_lisp (const lisp x);

void write\_seq (const lisp x);

}