**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: Иерархические списки

Вариант №8

|  |  |  |
| --- | --- | --- |
| Студентка гр. 5383 |  | Максимова Е.С. |
| Преподаватель |  | Фирсов М.А. |

Санкт-Петербург

**2016**

**зАДАЧА**

Вариант 8. Заменить в иерархическом списке все вхождения заданного элемента (атома) *x* на заданный элемент (атом) *y.*

**пОСТАНОВКА ЗАДАЧИ**

На вход подается любая последовательность символов заключенных в скобки, ввод заканчивается, только закрытием всех скобок, т.е. сколько скобок было открыто, столько должно быть и закрыто.

Иерархия списка указывается за счет скобочной структуры входных данных. Вложенные скобки свидетельствуют о том, что в списке содержится элемент, который также представляет собой список.

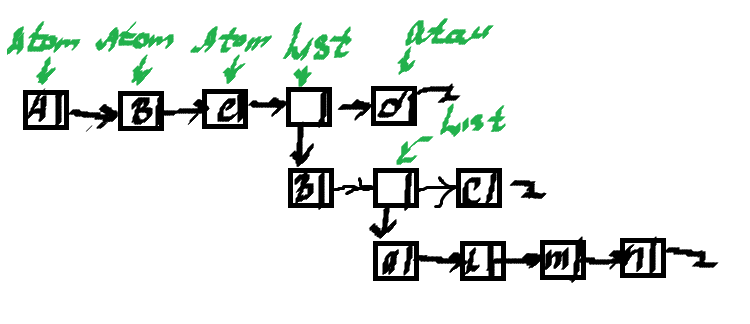
Пустой список задается в виде (). Преобразование пустого списка также пустой список.

**ПРИМЕР ИЕРАРХИЧЕСКОГО СПИСКА**

На вход подается последовательность: (a b c (b ( a l m n) c) d)

Данный иерархический список имеет три уровня. Уровни указываются вложенностью скобок.

*Графическое представление списка (рис. 1)*



**ОПИСАНИЕ СТРУКТУР ДАННЫХ**

В программе используются следующие структуры данных: линейный список, иерархический список, вектор (стандартный).

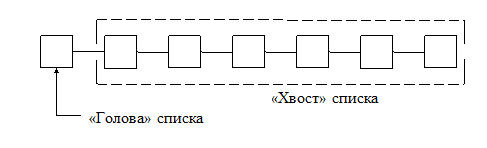
Линейный список:

Используется в иерархическом списке

<L\_List(El)> ::= <Null> | ( <Item(El)>, <Next(El)> )

<Item(El)> ::= <El>

<Next(El)> ::= <L\_List(El)>



Иерархический список:

<H\_List(El)> ::= <Atom(El)> | <L\_List(H\_List(El))>

<Atom(El)> ::= <El>

Другими словами, иерархический список, это либо атом, либо линейный список иерархических списков.

В данном варианте, атомом являются константы и переменные, а операция с аргументами – линейным списком.

**Описание алгоритма**

1. Ввод списка
2. Если список введен верно, последовательно сравнивать элементы с заданным значением и заменять на новое.
3. Перейти на следующий уровень.
4. Пока уровень не последний повторять пункты 2-3.
5. Вывести на экран преобразованный список.

**ОПИСАНИЕ ФУНКЦИЙ**

|  |  |  |
| --- | --- | --- |
| **Функция** | **Описание** | **Параметры** |
| lisp head(const lisp s); | Селектор. Базовая функция.  Возвращает“голову списка”.  Предусловие: список не пуст. | const lisp s –  входной параметр, указатель на голову списка. |
| lisp tail(const lisp s) | Селектор. Базовая функция.  Возвращает указатель на следующий элемент.  Предусловие: список не пуст. | const lisp s –  входной параметр, указатель на голову списка. |
| bool isAtom(const lisp s) | Определяет тип элемента списка  True- атом  False - список | const lisp s –  входной параметр, указатель на голову списка. |
| bool isNull(const lisp s) | Определяет пуст ли список  True- список пуст | const lisp s –  входной параметр, указатель на голову списка. |
| lisp cons(const lisp h, const lisp t) | Конструктор. Базовая функция. | Входные параметры  const lisp h – указатель на голову списка  const lisp t – указатель на хвост списка |
| lisp make\_atom  (const base x, int flag) | Формирует атом (базовый элемент списка) | Входные параметры  const base x – считываемое значение базового типа (символ из консоли)  int flag – переменная флажок, необходима для запоминания уровня рассматриваемого элемента в иерархическом списке |
| void destroy(lisp s) | Удаляет список | Входной параметр  lisp s – указатель на голову списка |
| void read\_lisp(lisp &y) | Ввод списка с консоли с обрамляющими его скобками | Транзитный параметр  lisp &y – указатель на голову списка |
| void read\_s\_expr(base prev, lisp &y) | Ввод и проверка верной расстановки скобок. | Входной параметр  base prev – ранее прочтанный символ  трианзитный параметр  lisp &y – указатель на голову списка |
| void read\_seq(lisp& y) | Ввод списка без обрамляющих его скобок | трианзитный параметр  lisp &y – указатель на голову списка |
| void write\_lisp(const lisp x) | Вывод списка со скобками | сonst lisp s – указатель на голову списка |
| void write\_seq(const lisp x) | Вывод списка без скобок | const lisp x– указатель на голову списка |
| lisp copy\_lisp(const lisp x) | Копирует список | const lisp x– указатель на голову списка |

**ПРИМЕР РАБОТЫ ПРОГРАММЫ**

**Введите список:**

**(hghy(juhtg(ki)hy)khew)**

**Введите элемент, который нужно заменить:**

**h**

**Введите элемент, на который нужно заменить:**

**0**

**Шаг 1. Был рассмотрен атом w**

**Шаг 2. Был рассмотрен атом e**

**Шаг 3. Был рассмотрен атом h и заменён на 0**

**Шаг 4. Был рассмотрен атом k**

**Шаг 5. Был рассмотрен атом y**

**Шаг 6. Был рассмотрен атом h и заменён на 0**

**Шаг 7. Был рассмотрен атом i**

**Шаг 8. Был рассмотрен атом k**

**Шаг 9. Был рассмотрен атом g**

**Шаг 10. Был рассмотрен атом t**

**Шаг 11. Был рассмотрен атом h и заменён на 0**

**Шаг 12. Был рассмотрен атом u**

**Шаг 13. Был рассмотрен атом j**

**Шаг 14. Был рассмотрен атом y**

**Шаг 15. Был рассмотрен атом h и заменён на 0**

**Шаг 16. Был рассмотрен атом g**

**Шаг 17. Был рассмотрен атом h и заменён на 0**

**Введенный список:**

**( h g h y ( j u h t g ( k i ) h y ) k h e w )**

**Преобразованный список:**

**( 0 g 0 y ( j u 0 t g ( k i ) 0 y ) k 0 e w )**

**ВЫВОД**

При выполнении лабораторной работы были получены практические навыки работы с таким абстрактным типом данных как иерархический список.

**ИСХОДНЫЙ КОД**

#include <iostream>

#include <cstdlib>

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

#include <locale>

using namespace std;

// объявление базового типа

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;

base x1, zam;

bool find\_zam = false;

int N = 0;

//.....................список функций...........................................

lisp head(const lisp s);

lisp tail(const lisp s);

lisp cons(const lisp h, const lisp t);

lisp make\_atom(const base x, int flag);

bool isAtom(const lisp s);

bool isNull(const lisp s);

void destroy(lisp s);

base getAtom(const lisp s);

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);

//...........................базовые функциию...................................

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";

system("Pause");

exit(1);

}

}

else

{

cout << "Error: Head(nil) \n";

system("Pause");

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

{

cout << "Error: Tail(atom) \n";

system("Pause");

exit(1);

}

}

else

{

cout << "Error: Tail(nil) \n";

system("Pause");

exit(1);

}

}

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

lisp cons(const lisp h, const lisp t)

// PreCondition: not isAtom (t)

{

lisp p = NULL;;

if (isAtom(t))

{

cout << "Error: Tail(nil) \n";

system("Pause");

exit(1);

}

else

{

p = new s\_expr;

if (p == NULL)

{

cout << "Memory not enough\n";

system("Pause");

exit(1);

}

else

{

p->tag = false;

p->node.pair.hd = h;

p->node.pair.tl = t;

return p;

}

}

}

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

lisp make\_atom(const base x, int flag)

{

lisp s = NULL;

int flag2 = 0;

s = new s\_expr;

s->tag = true;

if (x == x1)

{

s->node.atom = zam;

flag2 = 1;

find\_zam = true;

}

else

s->node.atom = x;

if (flag == 1)

{

N++;

if (flag2 == 0)

cout << "Шаг " << N << ". Был рассмотрен атом " << x << endl;

else

cout << "Шаг " << N << ". Был рассмотрен атом " << x << " и заменён на " << zam << endl;

}

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))

{

cout << "Error: getAtom(s) for !isAtom(s) \n";

system("Pause");

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 == ')')

{

cout << " ! List.Error 1 " << endl;

system("Pause");

exit(1);

}

else

{

if (prev != '(')

y = make\_atom(prev, 0);

else

read\_seq(y);

}

} //end read\_s\_expr

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

void read\_seq(lisp& y)

{

base x;

lisp p1, p2;

if (!(cin >> x))

{

cout << " ! List.Error 2 " << endl;

system("Pause");

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->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));

}

}

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

lisp copy\_lisp(const lisp x)

{

if (isNull(x))

return NULL;

else

{

if (isAtom(x))

return make\_atom(x->node.atom, 1);

else

return cons(copy\_lisp(head(x)), copy\_lisp(tail(x)));

}

}

//..........................главная функция.....................................

int main()

{

setlocale(LC\_ALL, "rus");

lisp s1 = NULL, s2 = NULL;

N = 0;

cout << "Введите список:" << endl;

read\_lisp(s1);

cout << "Введите элемент, который нужно заменить:" << endl;

cin >> x1;

cout << "Введите элемент, на который нужно заменить:" << endl;

cin >> zam;

s2 = copy\_lisp(s1);

//system("cls");

cout << "Введенный список: " << endl;

write\_lisp(s1);

cout << endl;

cout << "Преобразованный список: " << endl;

write\_lisp(s2);

cout << endl;

if (find\_zam == false) {

cout << "Элемент не найден!" << endl;

}

cout << endl;

destroy(s1);

destroy(s2);

//delete(s1);

//delete(s2);

system("pause");

return 0;

}

**ТЕСТИРОВАНИЕ**

|  |  |
| --- | --- |
| **Вводимый текст** | **Результат** |
| (a (b c) d)  x->a  y->z | (z (b c) d) |
| (a s d(b a) c)  x->a  y->k | (k s d(b k) c) |
| (a s d(b a) c)  x->a  y->0 | (0 s d(b 0) c) |
| (asd (ba) c)  x->a  y->k | (k s d(b k) c) |
| ) a s d(b a) c)  x->a  y->k | Error! |
| ()  Пустой список  x->a  y->k | () |
| ((((a))))  x->a  y->N | ((((N)))) |
| (asd (ba) c)  x->v  y->k | (asd (ba) c) |