**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. 6383 |  | Лавренкова Е.Л. |
| Преподаватель |  | Шолохова О.М. |

Санкт-Петербург

2017

**Цель работы.**

Практическое применение и освоение иерархических списков, структур данных, функций для представления информации в связанной памяти на языке программирования С++.

**Задание (вариант №4).**

Подсчитать число атомов в иерархическом списке; сформировать линейный список атомов, соответствующий порядку подсчета.

**Основные теоретические положения.**

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

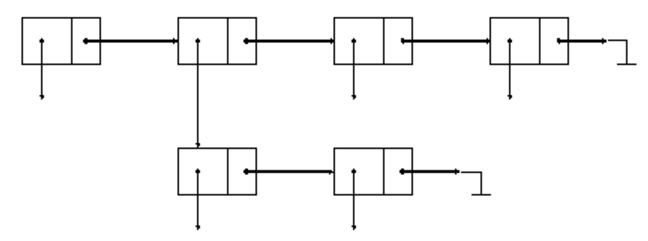
< Non\_null\_list(El) > ::= < Pair(El) >

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

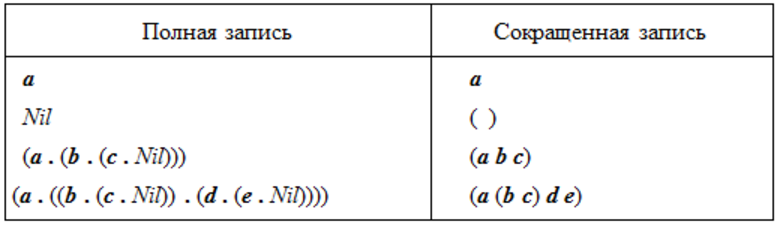
< Head\_l(El) > ::= < El >

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

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



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».

**Спецификация программы.**

*Назначение программы*.

Программа предназначена для подсчёта числа атомов в списке, а также для создания линейного списка атомов, соответствующего порядку подсчёта.

*Описание программы*.

Программа написана на языке С++. Входными данными для программы являются символы, задающие иерархический список, вводимые пользователем из файла. Предполагается, что данные, считываемые из файла, корректные. Выходными данными являются количество атомов в списке и линейный список, состоящий из этих атомов. Ограничения отсутствуют.

*Пример диалога с пользователем*.

Введен иерархический список:

(элементы иерархического списка)

Результаты анализа:

Количество атомов = количество атомов

Линейный список:

(элементы линейного списка)

Анализ успешно завершен...

Для продолжения нажмите любую клавишу . . .

*Реализация*.

Описание алгоритма:

Шаг 1.

Вызов функции analysis(const lisp x, elem& h, int & n). Если переданный ей иерархический список пуст, то указатель на первый элемент линейного списка получает значение NULL и анализ завершается; иначе переход к шагу 2.

Шаг 2.

Вызов функции work\_lisp(x,n,h,ah). Работу функции описывает Шаг 3. Присваивание значение NULL полю next текущего элемента линейного списка. Завершение анализа.

Шаг 3.

Если текущий элемент иерархического списка является атомом, то увеличение переменной n, отвечающей за количество атомов в списке, на 1, затем запись текущего элемента иерархического списка в линейный список; иначе вызов функции work\_seq(x,n,h,ah). Работу функции описывает Шаг 4.

Шаг 4.

Если иерархический список не пуст, то вызов функции work\_lisp(head (x),n,h,ah) (см. Шаг 3), а затем функции work\_seq(tail (x),n,h,ah) (см. Шаг 4).

Примечание: после завершения анализа иерархический и линейный списки удаляются.

Функции.

Для решения задачи используются следующие функции:

void analysis(const lisp x, elem& h, int & n); – Проверка иерархического списка на пустоту

void work\_lisp (const lisp x, int & n, elem& h, elem& ah); – Работа со списком с обрамляющими скобками

void work\_seq (const lisp x, int & n, elem& h, elem& ah); – Работа со списком без обрамляющих скобок

|  |  |  |  |
| --- | --- | --- | --- |
| *№* | *Тип* | *Имя* | *Назначение* |
| 1 | const lisp | x | Хранение иерархического списка |
| 2 | int & | n | Количество атомов в иерархическом списке |
| 3 | elem& | h | Указатель на первый элемент создающегося линейного списка |
| 4 | elem& | ah | Указатель на текущий элемент создающегося линейного списка |

Структуры данных

Иерархический список в данной лабораторной работе организован следующим образом:

Базовый тип атомов иерархического списка:

typedef char base;

|  |  |
| --- | --- |
| *Структура* | *Иллюстрация* |
| 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; | two\_ptr:  hd  tl  s\_expr:  tag  atom  hd  tl |
| struct List  {  base atom;  List \*next;  };  typedef List \*elem; | atom  next |

**Тестирование.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Набор тестовых данных** | | **Предполагаемые результаты, высчитанные вручную** | **Результаты выполнения программы** | **Линейный список** | **Сравнительный анализ** |
| **№** | **Данные** |
| 1 | ((((r)))) | 1 | 1 | r | Ошибок нет |
| 2 | (a s y) | 3 | 3 | asy | Ошибок нет |
| 3 | (d y (t I )o p) | 6 | 6 | dytlop | Ошибок нет |
| 4 | (((y I u p) k y n w r) k o) | 11 | 11 | ylupkynwrko | Ошибок нет |
| 5 | () | 0 | 0 | - | Ошибок нет |
| 6 | E | 1 | 1 | E | Ошибок нет |
| 7 | (e) | 1 | 1 | e | Ошибок нет |
| 8 | ((())) | 0 | 0 | ­- | Ошибок нет |
| 9 | ((a p r o () (r o (g h) v k) e n) h g o r) | 16 | 16 | aprorohvkenhgor | Ошибок нет |

**Выводы.**

В процессе выполнения работы были получены практические навыки по применению структур иерархических списков, структур данных на языке программирования С++.

**Приложение. Исходный код.**

#include <iostream>

#include <cstdlib>

#include <windows.h>

#include <fstream>

using namespace std;

typedef char base;

struct s\_expr;

struct two\_ptr {

s\_expr \*head;

s\_expr \*tail;

};

struct s\_expr {

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

union {

base atom;

two\_ptr pair;

} node;

};

struct List {

base atom;

List \*next;

};

typedef s\_expr \*lisp;

typedef List \*elem;

lisp head(const lisp s);

bool isAtom(const lisp s);

bool isNull(const lisp s);

lisp tail(const lisp s);

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

lisp make\_atom(const base x);

void destroy(lisp s);

base getAtom(const lisp s);

void read\_lisp(lisp& y, ifstream &infile);

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

void read\_seq(lisp& y, ifstream &infile);

void write\_lisp(const lisp x);

void write\_seq(const lisp x);

lisp copy\_lisp(const lisp x);

void analysis(const lisp x, elem& b, int & n);

void work\_lisp(const lisp x, int & n, elem& h, elem& ah);

void work\_seq(const lisp x, int & n, elem& h, elem& ah);

void output(elem& b, ofstream &outfile);

void destroy\_lin(elem b);

int main() {

SetConsoleCP(1251); // для вывода кирилицы

SetConsoleOutputCP(1251); // для вывода кирилицы

ifstream infile("input.txt");

ofstream outfile("output.txt");

lisp s;

elem b;

int n = 0;

// Считывание иерархического списка из файла

read\_lisp(s, infile);

cout << "Введен иерархический список: " << endl;

// Вывод списка на экран

write\_lisp(s);

// Анализ иерархического списка;

// s - иерархический список, b - линейный список, n - количество атомов

analysis(s, b, n);

cout << endl << "Результаты анализа: " << endl << "Количество атомов = " << n << endl << "Линейный список: " << endl;

outfile << "Результаты анализа: " << endl << "Количество атомов = " << n << endl << "Линейный список: " << endl;

// Вывод результатов анализа;

output(b, outfile);

cout << endl;

outfile << endl;

cout << "Анализ успешно завершен..." << endl;

destroy(s);

destroy\_lin(b);

system("pause");

return 0;

}

lisp head(const lisp s) {

if (s != NULL)

if (!isAtom(s))

return s->node.pair.head;

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)

{

if (s != NULL)

if (!isAtom(s))

return s->node.pair.tail;

else

{

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

exit(1);

}

else

{

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

exit(1);

}

}

lisp cons(const lisp h, const lisp 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.head = h;

p->node.pair.tail = 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;

};

}

base getAtom(const lisp s)

{

if (!isAtom(s))

{

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

exit(1);

}

else

return (s->node.atom);

}

void read\_lisp(lisp& y, ifstream &infile)

{

base x;

do

infile >> x;

while (x == ' ');

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

}

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

{

if (prev == ')')

{

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

exit(1);

}

else

if (prev != '(')

y = make\_atom(prev);

else

read\_seq(y, infile);

}

void read\_seq(lisp& y, ifstream &infile)

{

base x;

lisp p1, p2;

if (!(infile >> x))

{

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

exit(1);

}

else

{

while (x == ' ')

infile >> x;

if (x == ')')

y = NULL;

else

{

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

read\_seq(p2, infile);

y = cons(p1, p2);

}

}

}

// Процедура вывода списка с обрамляющими его скобками - write\_lisp,

// а без обрамляющих скобок - write\_seq

void write\_lisp(const lisp x)

{

if (isNull(x))

cout << " ()";

else

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));

}

}

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)));

}

void analysis(const lisp x, elem& h, int & n)

{

elem ah;

if (isNull(x))

h = NULL;

else

{

work\_lisp(x, n, h, ah);

ah->next = NULL;

}

}

void work\_lisp(const lisp x, int & n, elem& h, elem& ah)

{

if (isAtom(x))

{

// Подсчет атомов;

n++;

// Создание линейного списка;

if (n == 1)

{

ah = new List;

ah->atom = x->node.atom;

h = ah;

}

else

{

ah->next = new List;

if (n == 2)

h = ah;

ah = ah->next;

ah->atom = x->node.atom;

}

}

else

work\_seq(x, n, h, ah);

}

void work\_seq(const lisp x, int & n, elem& h, elem& ah)

{

if (!isNull(x))

{

work\_lisp(head(x), n, h, ah);

work\_seq(tail(x), n, h, ah);

}

}

void output(elem &b, ofstream &outfile)

{

if (b != NULL)

while (b != NULL)

{

outfile << b->atom << " ";

cout << b->atom << " ";

b = b->next;

}

else

cout << "()" << endl;

}

void destroy\_lin(elem b)

{

elem help;

while (b != NULL)

{

help = b->next;

delete b;

b = help;

}

delete b;

}