**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. 8381 |  | Звегинцева Е.Н. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2019

**Цель работы.**

Познакомиться с основными функциями создания и обработки иерархического списка.

**Постановка задачи.**

Задание №6.

Проверить иерархический список на наличие в нем заданного элемента (атома) *x*;

**Основные теоретические положения.**

Представление иерархического списка

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).

***d***

***e***

***a***

***b***

***c***

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, как показано в табл.1.

Таблица 1 - Примеры перехода от полной к сокращенной скобочной записи иерархических списков.

|  |  |
| --- | --- |
| Полная запись | Сокращенная запись |
| *a*  *Nil*  (*a* . (*b* . (*c* . *Nil*)))  (*a* . ((*b* . (*c* . *Nil*)) . (*d* . (*e* . *Nil*)))) | *a*  (  )  (*a b c*)  (*a* (*b c*) *d e*) |

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

**Выполнение работы.**

Создаем структуры данных *s\_expr*, в которой содержится два поля: *tag* (флаг: 1-атом, 0-узел) и *union node* (хранит в себe значение атома *atom* или структуру *two\_pair*), *two\_pair*, в которой храниться указатели на атом или узел.

Создаем функции:

* *read\_lisp()* - Пропускает пробелы и вызывает функция под номером 2. На вход подается указатель на список и название потока ввода
* *read\_s\_expr()* - Создает атомы и вызывает функцию с номер 3 для дальнейшей обработки строки. На вход подается последний символ, считанный с потока, указатель на список и название потока ввода.
* *read\_seq()* - Рекурсивная функция, которая обрабатывает строку и создает и скрепляет узлы между собой. На вход подается указатель на список и название потока ввода
* *lisp make\_atom()* - Создает структуру с атомом. На вход подается имя атом. Возвращает указатель, на созданную струтуру
* *lisp cons()* - Присоединяет узел к списку. На вход подается указатель на голову и хвост узла. Возвращает указатель на присоединенный узел.
* *write\_lisp()* - Выводит на экран список в виде атомов и узлов в виде скобок. На вход подается указатель на список и название потока ввода.
* *write\_seq()* - Выводит на экран список в виде атомов и узлов в виде скобок.Сама функция выводит непосредственно хвост узла. На вход подается указатель на список и название потока ввода.
* *isAtom()* - Проверяет является ли этот элемент списка атомом или узлом. На вход подается указатель на структуру. Возвращает 1 – если это атом, 0 – если узел.
* *lisp tail()* - Возвращает указатель на tail списка. На вход подается указатель на структуру s.
* *lisp head()* - Возвращает указатель на head списка. На вход подается указатель на структуру s.
* *isNull()* - Проверяет является ли список пустым. На вход подается указатель на структуру s. Возвращает 1 – если пуст, 0 – если не пуст.
* *Check()* - Возвращает 1- если элемент найдем, 0 – если не найден. На вход подается указатель на Иерархический список, Искомый Атом x, начальное значение результата поиска.

**Тестовые задания.**

|  |  |
| --- | --- |
| Входные данные | Исходные данные |
| ( a ( b ) ) |  |
| (x y z) |  |
| (1 2 3 (4) ) |  |
| ( d f g (f g h)) |  |
| Проверка на пустой файл |  |

**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

#include <iostream>

#include <cstdlib>

#include <fstream>

using namespace std;

struct s\_expr;

int c=0;

struct two\_ptr

{

s\_expr \*hd;

s\_expr \*tl;

} ; //end two\_ptr;

struct s\_expr

{

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

union

{

char 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 char x);

bool isAtom (const lisp s);

bool isNull (const lisp s);

void destroy (lisp s);

// функции ввода:

void read\_lisp ( lisp& y,istream & in=cin); // основная

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

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

// функции вывода:

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

void write\_seq (const lisp x);

void Space(int c) // создание отступов для вызовов/завершений фукнций

{

for (int j=1; j <=c; j++) cout <<"\_\_";

}

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"; exit(1);

}

}

else

{

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

return 0;

}

}

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";

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: Tail(nil) \n";

exit(1);

}

else

{

p = new s\_expr;

//cout << "node: " << p << endl << endl;

if ( p == NULL)

{

cerr << "Error: 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 char x)

{

//cout << "Current simbol: " << x << endl;

lisp s;

s = new s\_expr;

s -> tag = true;

s->node.atom = x;

//cout << "Addres simbol: " << s << endl;

return s;

}

// ввод списка с консоли

void read\_lisp ( lisp & y,istream & in)

{

char x;

do

{

in >> x;

}while (x==' ');

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

} //end read\_lisp

void read\_s\_expr (char prev, lisp & y,std::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,std::istream & in)

{

char x;

lisp p1, p2;

if (!(in >> x)) x=0;

if((in.eof()||(x==0)))

{

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

exit(1);

}else

{

//cout << "Current simbol:" << x << endl;

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,

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));

}

}

bool Check(const lisp HierarchList,char x,bool b)

{

if(isNull(HierarchList)||(b)) // Если список закончился или наше условие выполнено

{

return b;

}

else

{

if(HierarchList->tag) // Если текущий символ - атом

{

Space(c);

cout << "Последний считанный символ: " << HierarchList->node.atom << endl;

Space(c);

cout << "Проверка на совпадение с символом " << x << endl;

if (HierarchList->node.atom==x)//При совпадение с нашим элементом

{

b=true;

return b;

}

}

else

{

c++;

Space(c); // Создание отступов для наглядности глубины вызова функции

cout << "Вызов функции CHECK\_HEAD\_TO\_ATOM\_X" << endl;

b=Check(HierarchList->node.pair.hd,x,b);

Space(c); // Создание отступов для наглядности глубины вызова функции

cout << "Завершение функции CHECK\_HEAD\_TO\_ATOM\_X" << endl;

c--;

c++;

Space(c); // Создание отступов для наглядности глубины вызова функции

cout << "Вызов функции CHECK\_TAIL\_TO\_ATOM\_X" << endl;

b=Check(HierarchList->node.pair.tl,x,b);

Space(c); // Создание отступов для наглядности глубины вызова функции

c--;

cout << "Завершение функции CHECK\_TAIL\_TO\_ATOM\_X" << endl;

}

return b;

}

}

int main(void)

{

setlocale(0, "");

int i=0;

lisp HierarchList=NULL;

ifstream fin;

bool k=true;

while(k)

{

cout << endl << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

cout << "1. Ввод списка вручную\n" << "2. Ввод списка из файла\n"<< "3. Выход\n" << "Ваш выбор: ";

cin >> i;

while(cin.fail())// Проверка на ошибки ввода

{

cin.clear();

cin.sync();

cout << "Ошибка номера!\n";

cout << "Ваш выбор: ";

cin >> i;

}

switch(i)

{

case 1:

std::cout << "Введите список:" << std::endl;

read\_lisp(HierarchList);//Считываем с консоли, поток по умолчанию cin

break;

case 2:

fin.open("atom.txt");

if(!fin.is\_open())

{

cout << "Файл не открыт\n";

system("pause");

return 1;

}

read\_lisp (HierarchList,fin);//Считываем из файла указывая поток fin

fin.close();

break;

case 3:

k=false;

break;

}

if (!k) break;

cout << "Введенный список: " ;

write\_lisp (HierarchList);// Выводим список на экран

char x;

cout << endl << "Введите Атом x(искомый атом): ";

cin >> x;

cout << endl;

bool b=false;

Space(c); // Создание отступов для наглядности глубины вызова функции

cout << "Вызов функции CHECK\_ATOM\_X" << endl;

b=Check(HierarchList,x,b);

Space(c); // Создание отступов для наглядности глубины вызова функции

cout << "Завершение функции CHECK\_ATOM\_X" << endl;

if(b)

{

cout << endl << "Атом x присутствует в списке";

}

else

{

cout << endl << "Атом x не присутствует в списке";

}

}

cin.get();

cin.get();

return 0;

}