**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

|  |  |  |
| --- | --- | --- |
| Студент гр. 6304 |  | Виноградов К.А. |
| Преподаватель |  | Балтрашевич В.Э. |

Санкт-Петербург

2017

**зАДАЧА**

Вариант 3. 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.

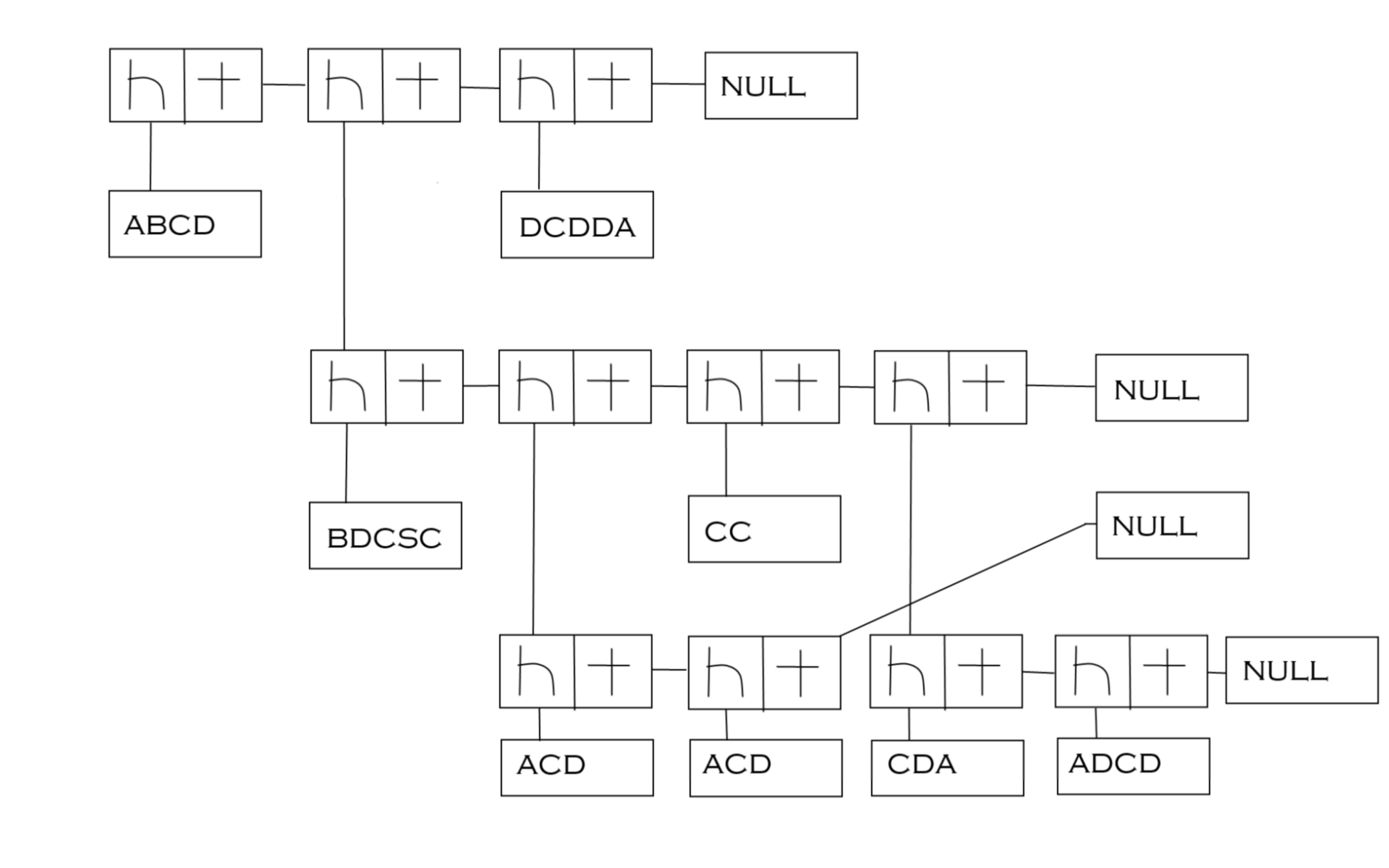
Иерархия списка указывается за счет скобочной структуры входных данных. Вложенные скобки свидетельствуют о том, что в списке содержится элемент, который также представляет собой список.

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

**ПРИМЕР ИЕРАРХИЧЕСКОГО СПИСКА**

На вход подается последовательность: ( abcd ( bdcsc( acd acd )cc( cda adcd ) ) dcdda)

Графическое представление списка (рис. 1)



**ОПИСАНИЕ СТРУКТУР ДАННЫХ**

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

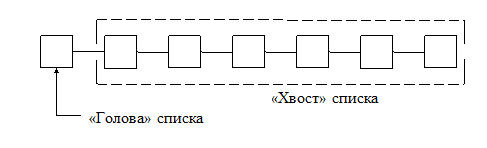
Линейный список:

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. Проанализировать список.
2. Если список введен верно, считывать содержимое списка, отделяя атомы от указателей.
3. Вывести список на экран.
4. Ввести заменяемый и заменяющие элементы.
5. Вывести на экран преобразованный список.

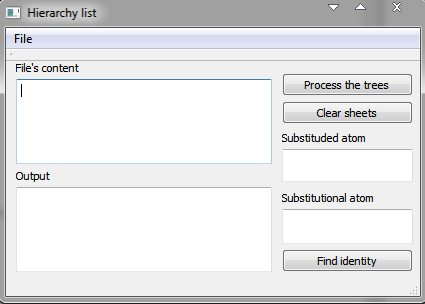
**ОПИСАНИЕ ФУНКЦИЙ**

|  |  |  |
| --- | --- | --- |
| **Функция** | **Описание** | **Параметры** |
| lisp head(const lisp s); | Возвращает голову непустого списка | * const lisp s – входной параметр, указатель на элемент списока. |
| lisp tail(const lisp s) | Возвращает указатель на следующий элемент непустого списка | * const lisp s – входной параметр, указатель на элемент списка. |
| bool isAtom(const lisp s) | Определяет тип элемента списка  True - атом  False - указатель | * const lisp s – входной параметр, указатель на элемент списка. |
| bool isNull(const lisp s) | Определяет пуст ли список  True - список пуст | * const lisp s – входной параметр, указатель на элемент списка. |
| lisp cons(const lisp h, const lisp t) | Конструктор. Базовая функция. | * const lisp h –указатель на голову списка * const lisp t –указатель на хвост списка |
| lisp make\_atom  (const base x) | Формирует атом (базовый элемент списка) | * const base x-считываемое значение базового типа |
| void eliminate(lisp s) | Удаляет список | * lisp s – указатель на элемент списка |
| void read\_lisp(lisp &list, char\* buffch) | Точка входа в рекурсивную функцию обработки списка | * lisp &list – указатель на элемент списка * char\* buffch – массив введенных данных |
| void read\_elem(char prev, lisp &list, base elem, char\* buffch, int i) | Чтение элемента списка. | * char prev – ранее прочтанный символ * lisp &list – указатель на элемент списка * char\* buffch – массив введенных данных * base elem – атом * int i - счетчик |
| void read\_seq(lisp& list, char\* buffch, int i) | Рекурсивная функция обработки списка | * lisp &list – указатель на элемент списка * char\* buffch – массив введенных данных * int i - счетчик |
| void write\_lisp(const lisp x, char\* arr, int i) | Точка входа в рекурсивную функцию вывода списка | * сonst lisp x – указатель на элемент списка * char\* arr – массив для сохранения данных * int i - счетчик |
| void write\_seq(const lisp x, char\* arr, int i) | Рекурсивная функция вывода списка | * const lisp x – указатель на элемент списка * char\* arr – массив для сохранения данных * int i - счетчик |
| int analyser(QString buffer, int length) | Функция-анализатор | * QString buffer – вводимые данные * int length – длина массива данных |
| int change\_atoms(const lisp x, const char \*first\_line, const char\* second\_line, int i) | Точка входа в рекурсивную функцию замены элементов | * const lisp x – указатель на элемент списка * const char\* first\_line – заменяемый атом * const char\* second\_line – заменяющий атом * int i - счетчик |
| int change\_atoms\_next(const lisp x, const char\* first\_line, const char\* second\_line, int i) | Рекурсивная функция замены элементов | * const lisp x – указатель на элемент списка * const char\* first\_line – заменяемый атом * const char\* second\_line – заменяющий атом * int i - счетчик |

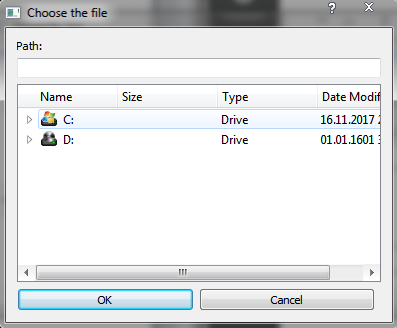
**ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС**

Для более удобной работы программы был создан пользовательский интерфейс:

* основное окно программы



* окно выбора файла



**ТЕСТИРОВАНИЕ**

|  |  |
| --- | --- |
| **Вводимый текст** | **Результат** |
| (a (b c) d)  Заменяемый элемент – a  Заменяющий элемент – z | (z(b c)d) |
| (a s d(b a) c)  Заменяемый элемент – a  Заменяющий элемент – k | (k s d(b k)c) |
| (a s d(b a) c)  Заменяемый элемент – a  Заменяющий элемент – 0 | (0 s d(b 0)c) |
| (asd (ba) c)  Заменяемый элемент – asd  Заменяющий элемент – mbk | (mbk(ba)c) |
| ) a s d(b a) c) | Error! |
| () | ( ) |
| ((((a))))  Заменяемый элемент – a  Заменяющий элемент – N | ((((N)))) |
| (asd (ba) c)  Заменяемый элемент – ba  Заменяющий элемент – nik | (asd(nik)c) |

**ВЫВОД**

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

**ИСХОДНЫЙ КОД**

* файл dialog.cpp

#include "dialog.h"

#include "ui\_dialog.h"

#include <QPushButton>

Dialog::**Dialog**(QWidget \*parent) :

QDialog(parent),

ui(new Ui::Dialog)

{

ui->setupUi(this);

model=new QFileSystemModel(this);

model->setFilter(QDir::AllEntries|QDir::NoDotAndDotDot|QDir::AllDirs);

model->setRootPath("");

ui->treeView->*setModel*(model);

connect(ui->ok, SIGNAL(clicked()),SLOT(on\_ok\_clicked()));

connect(ui->cancel, SIGNAL(clicked()), SLOT(close()));

connect(ui->treeView, SIGNAL(clicked(QModelIndex)),

SLOT(on\_treeView\_clicked(QModelIndex)));

}

Dialog::~***Dialog***()

{

delete ui;

}

void Dialog::**on\_ok\_clicked**()

{

emit FilePath(ui->lineEdit->text());

close();

}

void Dialog::**on\_cancel\_clicked**()

{

close();

}

void Dialog::**on\_treeView\_clicked**(const QModelIndex &index)

{

QFileInfo fileInfo=model->fileInfo(index);

ui->lineEdit->setText(fileInfo.absoluteFilePath());

}

* файл main.cpp

#include "mainwindow.h"

#include <QApplication>

#include <QPushButton>

using namespace std;

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}

* файл mainwindow.cpp

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <iostream>

#include <cstring>

#include <fstream>

#include <QFile>

#include <QDir>

#include "dialog.h"

#include <QTextStream>

#include <QMessageBox>

#include <QClipboard>

using namespace std;

QString buff;

int flag=0;

int creation\_flag=0;

MainWindow::**MainWindow**(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

}

MainWindow::~***MainWindow***()

{

delete ui;

}

typedef char\* base;

struct s\_expr;

struct two\_ptr

{

s\_expr \*hd;

s\_expr \*tl;

};

struct s\_expr

{

bool tag;

union

{

base atom;

two\_ptr pair;

} node;

};

typedef s\_expr \*lisp;

base x1, zam;

bool find\_zam = false;

int N = 0;

lisp hier\_list=NULL;

int **analyser**(QString buffer, int length);

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 **eliminate**(lisp s);

base **getAtom**(const lisp s);

void **read\_lisp**(lisp& list, char\* buffch, int length);

int **read\_elem**(char prev, lisp& list, base elem, char \*buffch, int i, int length);

int **read\_seq**(lisp& list, char \*buffch, int i, int length);

int **write\_lisp**(const lisp x, char \*arr, int i);

int **write\_seq**(const lisp x, char \*arr, int i);

int **change\_atoms**(const lisp x, const char\* first\_line, const char \*second\_line, int i);

int **change\_atoms\_next**(const lisp x, const char \*first\_line, const char \*second\_line, int i);

void **read\_lisp**(lisp &list, char \*buffch, int length)

{

int k=0;

bool fl1=false;

bool fl2=false;

int i=0;

int j=0;

base elem = new char;

do

{

if((buffch[i]!=' ')&&(buffch[i]!='(')&&(buffch[i]!=')'))

{

fl1=true;

elem[j]=buffch[i];

i++;

j++;

}

else if((buffch[i]=='(')||(buffch[i]==')'))

{

fl1=false;

fl2=true;

elem[j]=buffch[i];

i++;

j++;

}

else

{

if((fl1==true)||(fl2==true))

{

fl2=false;

fl1=false;

i++;

break;

}

else

i++;

}

}

while(((buffch[i]!='(')&&(buffch[i]!='\0')&&(buffch[i]!=')')&&(buffch[i-1]!='(')&&(buffch[i-1]!=')'))||

(buffch[i]==' '));

elem[j]='\0';

k=read\_elem(elem[j-1], list, elem, buffch, i, length);

if(i<k)

i=k;

j=0;

i++;

}

int **read\_elem**(char prev, lisp& list, base elem, char\* buffch, int i, int length)

{

int j=i;

if (prev == ')')

{

return i;

}

else

{

if (prev != '(')

list = make\_atom(elem);

else

{

j=read\_seq(list, buffch, j++, length);

if(i<j)

i=j;

}

}

return i;

}

int **read\_seq**(lisp& list, char \*buffch, int i, int length)

{

bool fl1=false;

bool fl2=false;

int p=0;

int k=0;

int j=0;

base elem = new char[length];

lisp part1;

lisp part2;

if (buffch[i]=='\0')

{

return i;

}

else

{

do

{

if((buffch[i]!=' ')&&(buffch[i]!='(')&&(buffch[i]!=')'))

{

fl1=true;

elem[j]=buffch[i];

i++;

j++;

}

else if((buffch[i]=='(')||(buffch[i]==')'))

{

fl1=false;

fl2=true;

elem[j]=buffch[i];

i++;

j++;

}

else

{

if((fl1==true)||(fl2==true))

{

fl2=false;

fl1=false;

i++;

break;

}

else

i++;

}

}

while(((buffch[i]!='(')&&(buffch[i]!='\0')&&(buffch[i]!=')')&&(buffch[i-1]!='(')&&(buffch[i-1]!=')'))||

(buffch[i]==' '));

elem[j]='\0';

if (elem[j-1] == ')'){

list = NULL;

return i;

}

else

{

p=read\_elem(elem[j-1], part1, elem, buffch, i, length);

if(i<p)

i=p;

k=read\_seq(part2, buffch, i, length);

if(i<k)

i=k;

list = cons(part1, part2);

}

}

return i;

}

lisp **make\_atom**(const base x)

{

lisp new\_one = NULL;

new\_one = new s\_expr;

new\_one->tag = true;

new\_one->node.atom = x;

return new\_one;

}

void MainWindow::**on\_actionExit\_triggered**()

{

if(creation\_flag!=0)

{

eliminate(hier\_list);

creation\_flag=0;

}

close();

}

void MainWindow::**on\_Enter\_clicked**()

{

char\* arr=new char[1000];

int i = 0;

QByteArray buffb = buff.toLatin1();

char\* buffch = buffb.data();

int length=buff.size();

if(analyser(buff, length)==1){

ui->textEdit->setText("Error");

return;

}

if(creation\_flag!=0)

{

eliminate(hier\_list);

creation\_flag=0;

}

creation\_flag=1;

read\_lisp(hier\_list, buffch, length);

write\_lisp(hier\_list, arr, i);

ui->textEdit->setText(arr);

}

int **analyser**(QString buffer, int length)

{

int i=0;

QChar open\_bracket;

int elem\_counter=0;

int\* temp=new int[10];

int lvl\_counter=0;

while(buffer[i]==' ')

i++;

if(buffer[i]!='(')

return 1;

temp[0]=0;

for(i=i; i<length; i++)

{

open\_bracket=buffer[i];

if((open\_bracket!='(')&&(open\_bracket!=')'))

elem\_counter++;

if(open\_bracket=='(')

{

temp[lvl\_counter]=elem\_counter;

lvl\_counter++;

elem\_counter=0;

}

if(open\_bracket==')')

{

lvl\_counter--;

elem\_counter=temp[lvl\_counter];

}

}

if(lvl\_counter!=0)

return 1;

return 0;

}

bool **isAtom**(const lisp s)

{

if (s == NULL)

return false;

else

return (s->tag);

}

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

{

lisp p = NULL;;

if (isAtom(t))

{

exit(1);

}

else

{

p = new s\_expr;

if (p == NULL)

{

exit(1);

}

else

{

p->tag = false;

p->node.pair.hd = h;

p->node.pair.tl = t;

return p;

}

}

}

lisp **head**(const lisp s)

{

if (s != NULL)

{

if (!isAtom(s))

return s->node.pair.hd;

else

{

exit(1);

}

}

else

{

exit(1);

}

}

bool **isNull**(const lisp s)

{

return s == NULL;

}

lisp **tail**(const lisp s)

{

if (s != NULL)

{

if (!isAtom(s))

return s->node.pair.tl;

else

{

exit(1);

}

}

else

{

exit(1);

}

}

void **eliminate**(lisp s)

{

if (s != NULL)

{

if (!isAtom(s))

{

eliminate(head(s));

eliminate(tail(s));

}

delete s;

s = NULL;

}

}

int **write\_lisp**(const lisp x, char\* arr, int i)

{

int j=0;

if (isNull(x)){

arr[i]='(';

arr[i+1]=' ';

arr[i+2]=')';

arr[i+3]='\0';

}

else

{

if (isAtom(x)){

while(x->node.atom[j]!='\0')

{

arr[i]=x->node.atom[j];

i++;

j++;

}

arr[i]=' ';

i++;

}

else

{

if((i-1>0)&&(arr[i-1]==' '))

i--;

arr[i]='(';

i++;

i=write\_seq(x, arr, i);

if((i-1>0)&&(arr[i-1]==' '))

i--;

arr[i]=')';

i++;

arr[i]='\0';

}

}

return i;

}

int **write\_seq**(const lisp x, char\* arr, int i)

{

if (!isNull(x))

{

i=write\_lisp(head(x), arr, i);

i=write\_seq(tail(x), arr, i);

}

return i;

}

int **change\_atoms**(const lisp x, const char \*first\_line, const char\* second\_line, int i)

{

int j=0;

if (isNull(x)){

return 1;

}

else

{

if (isAtom(x)){

if(strcmp(x->node.atom, first\_line)==0)

{

x->node.atom=new char[sizeof(second\_line)+1];

while(second\_line[j]!='\0')

{

x->node.atom[j]=second\_line[j];

j++;

i++;

}

x->node.atom[j]='\0';

}

}

else

{

i=change\_atoms\_next(x, first\_line, second\_line, i);

}

}

return i;

}

int **change\_atoms\_next**(const lisp x, const char\* first\_line, const char\* second\_line, int i)

{

if (!isNull(x))

{

i=change\_atoms(head(x), first\_line, second\_line, i);

i=change\_atoms\_next(tail(x), first\_line, second\_line, i);

}

return i;

}

void MainWindow::**on\_actionOpen\_file\_triggered**()

{

Dialog \*wnd = new Dialog(this);

wnd->show();

connect(wnd, SIGNAL(FilePath(QString)), this, SLOT(ReadToFile(QString)));

}

QString MainWindow::**ReadToFile**(const QString &FilePath)

{

QFile input(FilePath);

if(!input.*open*(QFile::ReadOnly|QFile::Text))

{

QMessageBox::information(this, "Error", "Path is not correct!");

return '\0';

}

QTextStream stream(&input);

int p=input.*size*();

if(p==0)

{

input.*close*();

return '\0';

}

buff=stream.readAll();

ui->textEdit\_2->setText(buff);

input.*close*();

flag=1;

return buff;

}

void MainWindow::**on\_pushButton\_clicked**()

{

if(creation\_flag!=0)

{

eliminate(hier\_list);

creation\_flag=0;

}

ui->textEdit->clear();

ui->textEdit\_3->clear();

ui->textEdit\_4->clear();

}

void MainWindow::**on\_pushButton\_2\_clicked**()

{

char\* arr=new char[1000];

int i=0;

QString chaged\_line=ui->textEdit\_3->toPlainText();

QString new\_line=ui->textEdit\_4->toPlainText();

QByteArray b\_changed\_line=chaged\_line.toLatin1();

QByteArray b\_new\_line=new\_line.toLatin1();

change\_atoms(hier\_list, b\_changed\_line, b\_new\_line, i);

write\_lisp(hier\_list, arr, i);

ui->textEdit->setText(arr);

}