```
/*
 * File:   Lista_TCP.h
 * Author: inf-coccodi-de
 *
 * Created on 12 febbraio 2009, 8.15
 */

#ifndef _LISTA_TCP_H
#define _LISTA_TCP_H

#include <stdio.h>
#include "addLib.h"
#define MAX_ETH 1500

/* ----------------------------------------------------------------------
  */

class Node
{
private:
    Node* next;
    Node* previous;
public:
    Node(Node*,Node*);
    Node();
    virtual ~Node();
    virtual void show();
    void set_key();
    void set_next(Node*);
    void set_previous(Node*);
    int get_key();
    Node* get_previous();
    Node* get_next();
};

/* ----------------------------------------------------------------------
  */

Node::Node(Node* newPrevious, Node* newNext)
{
    next = newNext;
    previous=newPrevious;
}

Node::Node()
{
    next = NULL;
    previous=NULL;
}

Node::~Node()
{
}

void Node::show()
{
    printf("precedente:%p__successivo:%p\n",previous ,next);
    printf("puntatore istanza corrente %p\n", this);
}

void Node::set_next(Node * n_nodo)
{
    next = n_nodo;
}

void Node::set_previous(Node* n_nodo)
{
    previous = n_nodo;
}

Node* Node::get_next()
{
    return (next);
}

Node* Node::get_previous()
{
    return (previous);
}

/* ---------------------------------------------------------------------------
  */

class Connection:public Node
{
private:
    Address* myAddr;
    int conn_id;
public:
    Connection(Address*,Node*,Node*,int);
    Connection(int);
    Connection(Address*,int);
    Connection(char*,int,int);
    ~Connection();
    void show();
    void setAddr(Address*);
    Address* getAddr();
    void setConn_id(int);
    int getConn_id();
    void invia(char*);
    char* ricevi();


};

/* ---------------------------------------------------------------------------
  */

Connection::Connection(Address* _a,Node* newPrevious, Node* newNext,int
  _conn_id)
:Node(newPrevious,newNext)
{
 myAddr=_a;
 conn_id=_conn_id;
}

Connection::Connection(Address* _a,int _conn_id)
:Node()
{
    myAddr=_a;
    conn_id=_conn_id;
}

Connection::Connection(char* _c,int _i,int _conn_id)
:Node()
{
    myAddr=new Address(_i,_c);
    conn_id=_conn_id;
}

Connection::Connection(int _conn_id):
Node()
{
    myAddr=new Address(8000,"0.0.0.0");
    conn_id=_conn_id;
}

Connection::~Connection()
{
    //Node::~Node();
/* DEBUG*/ printf ("---->distruttore Connection\n");

    shutdown(conn_id,SHUT_RDWR);
    delete(myAddr);
}

void Connection::show()
{
    Node::show();
    myAddr->stampaAdd();
    printf("\nConn_id: %d\n",conn_id);
}

void Connection::setAddr(Address* _Add)
{
    myAddr = _Add;
}

Address* Connection::getAddr()
{
    return (myAddr);
}

void Connection::setConn_id(int Replacement)
{
    conn_id = Replacement;
}

```

```cpp
int Connection::getConn_id()
{
    return (conn_id);
}

void Connection::invia(char* msg)
{
 int len_tx;
 int lenMsg;
 lenMsg=lenStr(msg); //inserisci in lenMsg la lunghezza del messaggio
 len_tx=send(conn_id,msg,lenMsg,0);//funzione che invia una stringa
 if(len_tx!=lenStr(msg)) //se la lunghezza del messaggio inviato effettivamente e' minore della lunghezza del messaggio originale
 {
     errore("send()=",len_tx);//gestione dell'errore in invio
 }
};

char* Connection::ricevi()
{
    int rx_len;
    char* buffer;//variabile che conterra' il messaggio ricevuto
    buffer=(char*)malloc(sizeof(char)*1501);
    rx_len=recv(conn_id,buffer,MAX_ETH,0);
     if(rx_len>0)//se il messaggio ricevuto ha almeno un carattere al'interno
    {
          fflush(stdout);
        *(buffer+(rx_len))='\0';//inserisci il valore di fine-stringa ad essa
          // printf("\n----->Interna: %s<------\n",buffer);
          //fflush(stdout);
    }
    else //altrimenti
    {
        //errore("recv()=",rx_len);//ritorna il genere di errore riscontrato
        return (NULL);
    }
    return (buffer); //ritorna la copia del messaggio ricevuto
}

/* ------------------------------------------------------------------ 
  */

class List; //prototipo di classe

/* ------------------------------------------------------------------ 
  */

class Iterator
{
private:
    List* myList;
    Node* current;
public:
    Iterator(List*,Node*);
    Iterator(List*);
    Iterator();
    ~Iterator();
    Node* getCurrent();
    void setCurrent(Node*);
    void moveFirst();
    void moveLast();
    void movePrevious();
    void moveNext();
    int isFirst();
    int isLast();
    void showCurrent();
};

/* ------------------------------------------------------------------------
  */

class List
{
private:
    Node* first;
    Node* last;
    void deleteList(Node*);
    void showList(Node*);
public:
    List();
    ~List();
    int is_empty();
    void show();
    Node* getFirst();
    Node* getLast();
    void addOnHead(Node*);
    void addOnTail(Node*);
    void removeFromHead();
    void removeFromTail();
    Iterator* createIterator();
    bool remove(Node*);
    bool exists(Node*);
};

/* ------------------------------------------------------------------------
  */

List::List()
{
    first = NULL;
    last=NULL;
}

List::~List()
{
    deleteList(first);
    first=NULL;
    last=NULL;
}

int List::is_empty()
{
    return ((first) ? 0 : 1);
}

void List::show()
{
    showList(first);
}

void List::deleteList(Node* a)
{
    if (a!=NULL)
    {
        deleteList(a->get_next());
        delete(a);
    }
}

void List::showList(Node* a)
{
    if (a)
    {
        a->show();
        showList(a->get_next());
    }
}

void List::addOnHead(Node* incoming)
{
    if(last==NULL)
    {
            incoming->set_next(NULL);
            incoming->set_previous(NULL);
            last=incoming;
            first=incoming;
    }
    else
    {
            first->set_previous(incoming);
            incoming->set_next(first);
            incoming->set_previous(NULL);
            first=incoming;
    }
}

void List::addOnTail(Node* incoming)
{
    if(first==NULL)
    {
            incoming->set_next(NULL);
            incoming->set_previous(NULL);
            last=incoming;
            first=incoming;
    }
    else
    {
            last->set_next(incoming);
```

```
                incoming->set_previous(last);
                incoming->set_next(NULL);
                last=incoming;
        }
}

Node* List::getFirst()
{
    return (first);
}

Node* List::getLast()
{
    return (last);
}

void List::removeFromHead()
{
    Node* nodo;
    nodo=first;
    if(first!=NULL)
    {
        if(nodo->get_next()==NULL)
        {
            delete(nodo);
            first=NULL;
            last=NULL;
        }
        else
        {
            nodo=nodo->get_next();
            delete(first);
            first=nodo;
            first->set_previous(NULL);
        }
    }
}

void List::removeFromTail()
{
    Node* nodo;
    nodo=last;
    if(last!=NULL)
    {
        if(nodo->get_previous()==NULL)
        {
            delete(nodo);
            first=NULL;
            last=NULL;
        }
        else
        {
            nodo=nodo->get_previous();
            delete(last);
            last=nodo;
            last->set_next(NULL);
        }
    }
}

 Iterator* List::createIterator()
 {
   return (new Iterator(this));
 }

bool List::remove(Node* nodo)
{
 Node   *curry,*next,*prev;

 for (curry=first;(curry);curry=curry->get_next())
   if (curry==nodo)
       {
    next = nodo->get_next();
    prev = nodo->get_previous();
    if (next)
      next->set_previous(prev);
    else
      last=prev;
    if(prev)
      prev->set_next(next);
    else
      first=next;
    delete(nodo);
    return true;
       }
       return false;
}

bool List::exists(Node* nodo)
{
 Node* curr;

 for(curr=first;curr;curr=curr->get_next())
    if(curr==nodo)
      return true;
 return false;
}
 /* -------------------------------------------------------------------------- 
   */

Iterator::Iterator(List* _l, Node* _n) {
    myList = _l;
    current = _n;
}

Iterator::Iterator(List* _l) {
    myList = _l;
    current = _l->getFirst();
}

Iterator::Iterator() {
    myList = new List();
    current = NULL;
}

Iterator::~Iterator() {

}

Node* Iterator::getCurrent() {
    return (current);
}

void Iterator::setCurrent(Node* _n) {
    current = _n;
}

void Iterator::moveFirst() {
    current = myList->getFirst();
};

void Iterator::moveLast() {
    current = myList->getLast();
}

void Iterator::movePrevious() {
    if ((myList->getLast() != NULL) && (current->get_previous() != NULL)) {
        current = current->get_previous();
    }
}

void Iterator::moveNext() {
    if ((myList->getLast() != NULL) && (current->get_next() != NULL)) {
        current = current->get_next();
    }
}

int Iterator::isFirst() {
    return ((current->get_previous()) ? 0 : 1);
}

int Iterator::isLast() {
    return ((current->get_next()) ? 0 : 1);
}

void Iterator::showCurrent() {
    current->show();
}

#endif  /* _LISTA_TCP_H */


```