SZÉCHENYI ISTVÁN EGYETEM
MŰSZAKI TUDOMÁNYI KAR

INFORMATIKA TANSZÉK

BSC FOKOZATÚ INFORMATIKUS MÉRNÖK SZAK

DIPLOMAMUNKA

– melléklet –

Nagyméretű tetraéderhálózatok hatékony kezelési módszereinek vizsgálata

– forráskódok –

Martin József
mérnök informatikus

Győr 2011.

Tartalomjegyzék

Forráskódok

Valamennyi modul forráskódja megtalálható a dolgozat nyomtatott mellékletében. Ugyanakkor a melléklet helytakarékossági okokból nem tartalmaz minden tetranet modul változatot; csak a V3Y változathoz tartozó tetranet.h és tetranet.c fájlokat nyomtattuk ki. Minden más változat forráskódja a CD-mellékleten kapott helyet.

main.c

```
/*
 *  main.c
 *
 *  Kozponti inditofajl.
 *  2010-2011 - Martin Jozsef
 */

#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <string.h>

#include "common.h"
#include "errors.h"
#include "tetranet.h"
#include "vector.h"
#include "testcase.h"

void printAll( tTetranet tn ) {
    tTetraRef tr;
    tetranet_iteratorInit( tn );
    while(( tr = tetranet_iteratorNext( tn ) ) != NULL_TETRA ) {
        printTetra( tn, tr );
    }
}

void selfTest( tTetranet tn ) {
//    test_massPointLocation( tn );
    test_delete( tn );
    test_explode( tn );
    test_delete( tn );
    test_alfa( tn );
    test_flow( tn );
//  printAll( tn );
}

void help() {
    printf( "Test software to check large tetrahedron networks.\n" );
    printf( "2010 - Martin Jozsef\n\n" );
    printf( "Usage: %s nas_file_name\n", glob_swName );
    exit( EXIT_FAILURE );
}

int main( int argc, char *argv[] ) {
    common_setGlobSwName( argv[0] );
    if( argc < 2 ) help();
    common_setGlobSwDate();
    common_setGlobInputFile( argv[1] );

    startClock();
    stopClock( "start" );

    startClock();
    tTetranet myTNet;
    myTNet = tetranet_new();
    tetranet_init( myTNet, argv[1] );
    stopClock( "init" );

    selfTest( myTNet );

    startClock();
    tetranet_free( myTNet );
    stopClock( "free" );
    printf( "\n" );
    return 0;
}
```

tetranet.h (v3y)

```
/*
 *  tetranet.h
 *
 *  A tetraederhalozat leirasa es alapmuveletei. Valtozat: V3Y
 *  2010-2011 - Martin Jozsef
 */

#ifndef TETRANET_H_
#define TETRANET_H_

/* -------------------------------------------------------------------------------------
 *  tipusdefiniciok
 *
 *  a megvalositott adatszerkezet fuggvenyeben modosulhatnak
 * -------------------------------------------------------------------------------------*/

#include "common.h"
#include "vector.h"
typedef vector tPoint;
typedef unsigned long int tPointRef;
typedef unsigned long int tTetraRef;
typedef unsigned int tSideIndex; /* 0..3 */

#define N_STATE 3

/// konstans az ervenytelen / nem letezo pont jelzesere
#define NULL_POINT 0

/// konstans az ervenytelen / nem letezo tetraeder jelzesere
#define NULL_TETRA 0

// tipusok a dinamikus tombok atlathatobb definialasahoz
// todo lehetne egyszerübben, pl: double (*sideArea)[4]
typedef tTetraRef tSideNext[4];
typedef tPointRef tVertices[4];
typedef double    tSideArea[4];
typedef vector    tSideNormVect[4];
typedef int       tSideType[4];
typedef double    tStates[N_STATE];

/**
 *  tFreeTetra
 *   ilyen elemekbol allo lancolt listaban taroljuk a szabad helyeket.
 *   - A lancban a referenciak sorrendje kotelezoen novekvo!!!
 *   - Mindig van legalabb egy eleme, ami az utolso (lastTetraRef) utani indexet tartalmazza.
 */
typedef struct _tFreeTetra{
    tTetraRef ref;
    struct _tFreeTetra *next;
} tFreeTetra;

typedef struct {
    // Dinamikus tombok a tetraederek adatainak kezelesere
    // pontok koordinatai
    tPoint        *points;
    // alkoto pontok adatai
    tVertices     *vertices;
    // hatarolo oldalak adatai
    tSideArea     *sideArea;
    tSideNormVect *sideNormVect;
    tSideType     *sideType;
    tSideNext     *sideNext;
    // magara a tetraederre vonatkozo adatok
    double        *volume;
    tStates       *states;
    tPoint        *massPoint;

    // szamossagtarolas
    tPointRef      maxPointRef;    // a tomb utolso cimezheto helye
    tTetraRef      maxTetraRef;    // a tomb utolso cimezheto helye
    tPointRef      lastPointRef;   // az utolso hasznalt elem indexe
    tTetraRef      lastTetraRef;   // az utolso hasznalt elem indexe
    unsigned long  numberOfPoints;
    unsigned long  numberOfTetras; // a tetraederek szama;

    // az elso szabad elem indexe
//    tTetraRef      firstFreeTetraRef;

    // a bejaro aktualis helyzete
    tTetraRef      iteratorPos;
//    void          *iterator;

    // adott ponthoz tartozo tetraederek keresesehez
    void          *atVertex;
    void          *nearestp;

    // a szabad helyek listajanak kezdocime
    tFreeTetra    *freeTetra;
    // a szabad helyek listajanak utolso elemenek cime
    tFreeTetra    *lastFreeTetra;
} tTetranetDescriptor;

typedef tTetranetDescriptor *tTetranet;

/* ---------------------------------------------------------------------------
 *  fuggvenyek
 *
 *  definiciojuk allando, fuggetlen az adatszerkezettol.
 *  a fentebb definialt tipusokat hasznaljak parameterul es visszeteresi tipusul
 * ---------------------------------------------------------------------------*/

/*
 *  felepites / bovites / torles
 */

/// ures halozatleiro keszitese
tTetranet tetranet_new( );

/// a teljes halozat inicializalasa bemeno adathalmazzal
void      tetranet_init( tTetranet tn, char *filename );

/// egy uj pont hozzaadasa a pontracshoz
tPointRef tetranet_insertPoint( tTetranet tn, tPoint p );

/// egy tetraeder hozzaadasa a meglevo halozathoz
tTetraRef tetranet_insertTetra( tTetranet tn, tPointRef pr0, tPointRef pr1, tPointRef pr2,
tPointRef pr3 );

/// egy pont eltavolitasa a meglevo halozatbol
void      tetranet_delPoint( tTetranet tn, tPointRef pr );

/// egy tetraeder eltavolitasa a meglevo halozatbol
void      tetranet_delTetra( tTetranet tn, tTetraRef tr );

/*
 *  getterek: informacio kinyerese a halozatbol
 */

/// pontadatok lekerdezese a pont indexe alapjan
tPoint    tetranet_getPoint( tTetranet tn, tPointRef pr );

/// csucsok indexenek lekerdezese
tPointRef tetranet_getVertex( tTetranet tn, tTetraRef tr, unsigned vi );

/// tetraeder terfogata
double    tetranet_getTetraVolume( tTetranet tn, tTetraRef tr );

/// tetraeder sulypontja
tPoint    tetranet_getTetraMassPoint( tTetranet tn, tTetraRef tr );

/// az allapotvektor sti-edik eleme
double    tetranet_getState( tTetranet tn, tTetraRef tr, unsigned int sti );

/// oldalszomszed tetraeder
```

```
tTetraRef tetranet_getSideNext( tTetranet tn, tTetraRef tr, tSideIndex si );

/// oldal terulete
double    tetranet_getSideArea( tTetranet tn, tTetraRef tr, tSideIndex si );

/// oldal kifele mutato normalvektora
vector    tetranet_getSideNormalVector( tTetranet tn, tTetraRef tr, tSideIndex si );

/// megkeresi, hogy az adott pont melyik teraederben van
tTetraRef tetranet_getPointLocation( tTetranet tn, tPoint p );

/// utolso hasznalt tetraeder ref
tTetraRef tetranet_getLastTetraRef( tTetranet tn );

/// utolso hasznalt pont ref
tPointRef tetranet_getLastPointRef( tTetranet tn );

/// a tetraederek szama
unsigned long tetranet_getNumberOfTetras( tTetranet tn );

/// a pontok szama
unsigned long tetranet_getNumberOfPoints( tTetranet tn );

/*
 *  setterek: adatmodositas a halozatban
 */

/// az allapotvektor sti-edik elemenek beallitasa
void      tetranet_setState( tTetranet tn, tTetraRef tr, unsigned int sti, double value );

/// az szomszedossag beallitasa
/**
 ez a setter itt nem elegans, mert nem szabad user altal hivni.
 szukseges megis, hogy a neighbours modul adatszerkezettol fuggetlenul irni tudja a
szomszedossagi viszonyokat.
 tombos megoldas eseten a teljes tomb kikerulhetne a neighboursba, es akkor nem kellene,
 de listas esetben a szomszedossag a tetraeder strukrura egy eleme, igy nem.
*/
void      tetranet_setSideNext( tTetranet tn, tTetraRef tr, tSideIndex si, tTetraRef neighbour
);


/*
 *  iteratorok: tetrraederek sorozatat adjak vissza
 */

/// tetraederhalo bejarasanak inditasa
void      tetranet_iteratorInit( tTetranet tn );

/// a bejaras soran következö tetraeder
tTetraRef tetranet_iteratorNext( tTetranet tn );

/// adott ponthoz tartozo tetraederek lekerdezesenek kezdese
bool ( *tetranet_atVertexInit )( tTetranet tn, tPointRef pr );

/// csak az init utan: adott ponthoz tartozo tetraederek kozul a kovetkezo
tTetraRef( *tetranet_atVertexNext )( tTetranet tn );

/// a teljes tetranet altal foglalt memoria felszabaditasa, a halozat törlese
void tetranet_free( tTetranet tn );

// csak teszteleshez
void printTetra( tTetranet tn, tTetraRef tr );
void printNet( tTetranet tn );

#endif /* TETRANET_H_ */
```

tetranet.c (v3y)

```
/*
 *  tetranet.c
 *
 *  A tetraederhalozat leirasa es alapmuveletei. Valtozat: V3Y
 *  2010-2011 - Martin Jozsef
 */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "tetranet.h"
#include "neighbour.h"
#include "errors.h"
#include "common.h"
#include "nasreader.h"
#include "atvertex.h"
#include "nearestp.h"

typedef struct _tIterator {
    bool      active;
    tTetraRef pos;
    tFreeTetra *nextFree;
} tIterator;

/**
 * novekvo sorrendbe rakja a csucsok indexeit
 * @param p pontnegyes, itt adodik vissza a rendezett halmaz is
 */
void sortVertices( tPointRef p[4] ) {
    tPointRef temp;
#define CHECK(i, j) {if( p[i] > p[j] ){temp = p[i]; p[i] = p[j]; p[j] = temp;}}
    CHECK( 0, 3 )
    CHECK( 1, 2 )
    CHECK( 0, 1 )
    CHECK( 2, 3 )
    CHECK( 1, 2 )
}

void addTetra( tTetranet tn, tTetraRef tr, tPointRef vertx[4] ) {
    int k = 0;
    double len = 0;
    double dotP = 0;

    // ideiglenes valtozok az oldalcsucsindexek tarolasara
    // d az oldallal szemközti csucs
    vector a, b, c, d;
    vector n;

    // a pontok indexei növekvö sorrendben
    sortVertices( vertx );

    // a csucsok tarolasa
    for( k = 0; k <= 3; k++ ) {
        tn->vertices[tr][k] = vertx[k];
    }

    // az oldalakhoz tartozo adatok szamitasa
    for( k = 0; k <= 3; k++ ) {
        // kivalasztjuk az k-adik oldalhoz tartozo pontokat; eredetileg fv: getSidePoints
        switch( k ) {
        case 0:
            a = tn->points[vertx[1]];
            b = tn->points[vertx[2]];
            c = tn->points[vertx[3]];
            d = tn->points[vertx[0]];
            break;
        case 1:
            a = tn->points[vertx[0]];
            b = tn->points[vertx[2]];
            c = tn->points[vertx[3]];
            d = tn->points[vertx[1]];
            break;
        case 2:
            a = tn->points[vertx[0]];
            b = tn->points[vertx[1]];
            c = tn->points[vertx[3]];
            d = tn->points[vertx[2]];
            break;
        case 3:
            a = tn->points[vertx[0]];
            b = tn->points[vertx[1]];
            c = tn->points[vertx[2]];
            d = tn->points[vertx[3]];
            break;
        default:
            exitText( "Index failed by getSidePoints." );
        }

        // oldal normalvektora
        n = normalOfPlane( a, b, c );

        // kifele mutasson:
        // ha az AD vektor es a normalvektor skalaris szorzata pozitiv, akkor azonos terfelbe
mutatnak
        // TODO: ha pont nulla, akkor ez egy hibas (egysiku) tetraeder. itt lehetne ezt jol
ellenorizni
        dotP = dotProduct( vector_diff( d, a ), n );

        if( dotP > 0 ) {
            n = negativeVector( n );
        } else {
            dotP = - dotP;
        }
        if( dotP <= EPS ) {
            exitText( "I found a 2D tetrahedron." );
        }

        len = vector_length( n );

        // egysegnyi hosszu kifele mutato normalvektor
        tn->sideNormVect[tr][k] = vector_constMult( n, 1 / len );

        // a terulet a keresztszorzat hosszanak a fele
        tn->sideArea[tr][k] = len / 2;

        // az oldal tipusa -- tovabbi informaciok hianyaban egyelore 0
        tn->sideType[tr][k] = 0;

        /* hogy ezt a tömböt is inicializáljuk -
         * különben csak az elso iraskor foglalodik tenyleges fizikai memoria
         */
        tn->states[tr][N_STATE - 1] = 0.0;

        // ki kell kinullaznunk a szomszedot.
        tn->sideNext[tr][k] = NULL_TETRA;
    }

    // Terfogat = az oldalvektorok vegyes szorzatanak hatodresze
    // CSAK az oldalbeallitasok utan hivhato, mert az abcd vektorok utolso allapotat hasznalja
    double volume = tripleProduct(
                        vector_diff( b, a ),
                        vector_diff( c, a ),
                        vector_diff( d, a ) ) / 6.0;
    if( volume < 0 ) {
        tn->volume[tr] = -volume;
    } else {
        tn->volume[tr] = volume;
    }


    // tomegkozeppont
    tn->massPoint[tr] = massPoint( a, b, c, d );

    // tetraederek szama
    ++( tn->numberOfTetras );
}

bool isPointInTetra( tTetranet tn, tTetraRef tr, tPoint p ) {
    vector ap = vector_diff( p, tetranet_getPoint( tn, tetranet_getVertex( tn, tr, 0 ) ) );
    return ( dotProduct( ap, tetranet_getSideNormalVector( tn, tr, 1 ) ) < 0 ) &&
           ( dotProduct( ap, tetranet_getSideNormalVector( tn, tr, 2 ) ) < 0 ) &&
           ( dotProduct( ap, tetranet_getSideNormalVector( tn, tr, 3 ) ) < 0 ) &&
           ( dotProduct(
                 vector_diff( p, tetranet_getPoint( tn, tetranet_getVertex( tn, tr, 1 ) ) ),
                 tetranet_getSideNormalVector( tn, tr, 0 ) )
             < 0 );
}


/* **********************************
 *   Interface fuggvenyek
 * **********************************/

tTetranet tetranet_new( ) {
    tTetranet t = malloc( sizeof( tTetranetDescriptor ) );
    memset( t, '\0', sizeof( tTetranetDescriptor ) );
//    t->iterator = malloc( sizeof( tIterator ) );
//    (( tIterator * )t->iterator )->active = FALSE;
    return t;
}

void tetranet_init( tTetranet tn, char *filename ) {
    FILE      *iniFile;
    tPoint     tempPoint;
    tPointRef  tempTetra[4];
    unsigned long i;

    iniFile = fopen( filename, "r" );
    if( iniFile == NULL ) {
        exitText( "?FILE NOT FOUND ERROR\nREADY." );
    }

    tn->lastPointRef = 0;
    tn->lastTetraRef = 0;

    tn->maxPointRef = nasreader_getPointNr( iniFile );
    tn->maxTetraRef = nasreader_getTetraNr( iniFile );

    tn->numberOfPoints = 0;
    tn->numberOfTetras = 0;

    // TODO: eleve több helyet foglani, a finomitasokhoz, pl.: +10%
    unsigned long num = tn->maxPointRef + 1;
    tn->points       = malloc( num * sizeof( tPoint ) );

    // TODO: eleve több helyet foglani, a finomitasokhoz, pl.: +10%
    num = tn->maxTetraRef + 1;
    tn->vertices     = malloc( num * sizeof( tVertices ) );
    tn->sideArea     = malloc( num * sizeof( tSideArea ) );
    tn->sideNormVect = malloc( num * sizeof( tSideNormVect ) );
    tn->sideType     = malloc( num * sizeof( tSideType ) );
    tn->sideNext     = malloc( num * sizeof( tSideNext ) );
    tn->volume       = malloc( num * sizeof( double ) );
    tn->states       = malloc( num * sizeof( tStates ) );
    tn->massPoint    = malloc( num * sizeof( tPoint ) );

    // pontok olvasasa fajlbol
    i = 0;
    nasreader_readFirstPoint( iniFile, &tempPoint );
    do {
        ++i;
        tn->points[i] = tempPoint;
    } while( nasreader_readNextPoint( iniFile, &tempPoint ) );
    tn->lastPointRef = i;
    tn->numberOfPoints = i;

    // tetraederek olvasasa fajlbol
    i = 0;
    nasreader_readFirstTetra( iniFile, tempTetra );
    do {
        ++i;
        addTetra( tn, i, tempTetra );
    } while( nasreader_readNextTetra( iniFile, tempTetra ) );
    // az utolso ervenyes index tarolasa
    tn->lastTetraRef = i;
    // inicializalom a szabad elemek lancat:
    // az egyetlen lancszeme az utolso elem utan mutat
    tFreeTetra *tmp = malloc( sizeof( tFreeTetra ) );
    tmp->next  = NULL;
    tmp->ref = i + 1;
    tn->freeTetra = tmp;
    // hack: eltarolom ugy is, mint az utolso lancelem cimet
    tn->lastFreeTetra = tmp;
    // nem kell mar tobbet a file
    fclose( iniFile );

    // a teljes halozatra vonatkozo beallitasok
    neighbours_update( tn );

    atVertex_update( tn );
    tetranet_atVertexInit = &atVertex_init;
    tetranet_atVertexNext = &atVertex_next;

    nearestp_update( tn );
}

inline bool isTheSamePoint( tPoint p1, tPoint p2 ) {
    return (( p2.x - p1.x ) * ( p2.x - p1.x ) +
            ( p2.y - p1.y ) * ( p2.y - p1.y ) +
            ( p2.z - p1.z ) * ( p2.z - p1.z ) ) < EPS;
}

tPointRef tetranet_insertPoint( tTetranet tn, tPoint p ) {
    tPointRef k = nearestp_search( tn, p );
    if( isTheSamePoint( p, tn->points[k] ) ) {
        return k;
    } else {
        if( tn->lastPointRef >= tn->maxPointRef ) {
            tn->maxPointRef = tn->maxPointRef * 2;
            tn->points = realloc( tn->points, ( tn->maxPointRef + 1 ) * sizeof( tPoint ) );
            if( tn->points == NULL ) {
                exitText( "Realloc points : error." );
            }
        }
        ++( tn->lastPointRef );
        ++( tn->numberOfPoints );
        tn->points[tn->lastPointRef] = p;
        nearestp_addPoint( tn, tn->lastPointRef );
        return tn->lastPointRef;
    }
}

void      tetranet_delPoint( tTetranet tn, tPointRef pr ) {
    // semmi, nem eri meg a macerat. igy viszont memoriazabalas. TODO
}

tTetraRef tetranet_insertTetra( tTetranet tn, tPointRef pr0, tPointRef pr1, tPointRef pr2,
tPointRef pr3 ) {
    if( tn->freeTetra->ref >= tn->maxTetraRef ) {
        unsigned long num;
        tn->maxTetraRef = tn->maxTetraRef * 2;  // TODO atgondolni, hogy a duplazas nem eros-e
egy kicsit
        num = tn->maxTetraRef + 1;
        tn->vertices     = realloc( tn->vertices,     num * sizeof( tVertices ) );
        if( tn->vertices == NULL )
            exitText( "Realloc vertices : error." );
        tn->sideArea     = realloc( tn->sideArea,     num * sizeof( tSideArea ) );
        if( tn->sideArea == NULL )
            exitText( "Realloc sideArea : error." );
        tn->sideNormVect = realloc( tn->sideNormVect, num * sizeof( tSideNormVect ) );
        if( tn->sideNormVect == NULL )
            exitText( "Realloc sideNormVect : error." );
        tn->sideType     = realloc( tn->sideType,     num * sizeof( tSideType ) );
        if( tn->sideType == NULL )
            exitText( "Realloc sideType : error." );
        tn->sideNext     = realloc( tn->sideNext,     num * sizeof( tSideNext ) );
        if( tn->sideNext == NULL )
            exitText( "Realloc sideNext : error." );
        tn->volume       = realloc( tn->volume,       num * sizeof( double ) );
        if( tn->volume == NULL )
```

```
            exitText( "Realloc volume : error." );
        tn->states       = realloc( tn->states,       num * sizeof( tStates ) );
        if( tn->states == NULL )
            exitText( "Realloc massPoint : error." );
        tn->massPoint    = realloc( tn->massPoint,    num * sizeof( tPoint ) );
        if( tn->massPoint == NULL )
            exitText( "Realloc tetras : error." );
    }
    tPointRef vertx[4];
    vertx[0] = pr0;
    vertx[1] = pr1;
    vertx[2] = pr2;
    vertx[3] = pr3;

    tTetraRef newRef = tn->freeTetra->ref;

    addTetra( tn, newRef, vertx );

    // elobb atvertex, csak utana neighbours, mert utobbi elobbit hasznalja !!!
    atVertex_insert( tn, newRef );
    neighbours_insert( tn, newRef );

    // hol lesz az uj utolso elem?
    if( tn->lastTetraRef < newRef ) {
        tn->lastTetraRef = newRef;
    }

    //hol lesz a kovetkezo szabad hely?
    if( tn->freeTetra->next == NULL ) {
        // ha ez volt az utolso a szabad elemek listajaban
        tn->freeTetra->ref = tn->lastTetraRef + 1;
    } else {
        tFreeTetra *tmp = tn->freeTetra->next;
        free( tn->freeTetra );
        tn->freeTetra = tmp;
    }

    return newRef;
}

void      tetranet_delTetra( tTetranet tn, tTetraRef tr ) {
    // ervenytelenitjuk, azaz megjeloljuk ures helykent
    tn->volume[tr] = -1;

    // feljegyezzuk a szabad helyek listajaba
    /*  a lista a 3y valtozatban nem rendezett.
        ezert nem is hasznalahato bejarashoz, viszont gyors */

    if( tr == tn->lastTetraRef ) {
        // keressük meg az utolso lancszemet, es irjuk at az ujra:
        tn->lastFreeTetra->ref = tr;
        // valtozik a lastTetraRef is, az utolo ervenyes elemre:
        do {
            --( tn->lastTetraRef );
        } while( tn->volume[tn->lastTetraRef] < 0 );
    } else {
        // ez lesz a lista elso eleme:
        tFreeTetra *tmp = tn->freeTetra;
        tFreeTetra *newFreeTetra = malloc( sizeof( tFreeTetra ) );
        newFreeTetra->ref = tr;
        newFreeTetra->next = tmp;
        // modositjuk a beugrasi pontot is!
        tn->freeTetra = newFreeTetra;
    }

    // toroljuk a szomszednyilvantartasbol es az atvertex-bol
    neighbours_delete( tn, tr );
    atVertex_delete( tn, tr );
    // tetraederek szama
    --( tn->numberOfTetras );
}

tPoint    tetranet_getPoint( tTetranet tn, tPointRef pr ) {
    return tn->points[pr];
}
```

```
tPointRef tetranet_getVertex( tTetranet tn, tTetraRef tr, unsigned vi ) {
    return tn->vertices[tr][vi];
}

double    tetranet_getTetraVolume( tTetranet tn, tTetraRef tr ) {
    return tn->volume[tr];
}

tPoint    tetranet_getTetraMassPoint( tTetranet tn, tTetraRef tr ) {
    return tn->massPoint[tr];
}

double    tetranet_getState( tTetranet tn, tTetraRef tr, unsigned int sti ) {
    return tn->states[tr][sti];
}

void      tetranet_setState( tTetranet tn, tTetraRef tr, unsigned int sti, double value ) {
    tn->states[tr][sti] = value;
}

tTetraRef tetranet_getSideNext( tTetranet tn, tTetraRef tr, tSideIndex si ) {
    return tn->sideNext[tr][si];
}

void      tetranet_setSideNext( tTetranet tn, tTetraRef tr, tSideIndex si, tTetraRef nb ) {
    tn->sideNext[tr][si] = nb;
}

double    tetranet_getSideArea( tTetranet tn, tTetraRef tr, tSideIndex si ) {
    return tn->sideArea[tr][si];
}

vector    tetranet_getSideNormalVector( tTetranet tn, tTetraRef tr, tSideIndex si ) {
    return tn->sideNormVect[tr][si];
}

void      tetranet_iteratorInit( tTetranet tn ) {
    tn->iteratorPos = 0;
}

tTetraRef tetranet_iteratorNext( tTetranet tn ) {
    if( tn->iteratorPos >= tn->lastTetraRef ) {
        return NULL_TETRA;
    } else {
        do {
            ++( tn->iteratorPos );
        } while( tetranet_getTetraVolume( tn, tn->iteratorPos ) < 0 );
        return( tn->iteratorPos );
    }
}

tTetraRef tetranet_getPointLocation( tTetranet tn, tPoint p ) {
    tTetraRef ntr, xtr;
    tSideIndex k;
    tPointRef npr = nearestp_search( tn, p );
    atVertex_init( tn, npr );
    while(( ntr = atVertex_next( tn ) ) != NULL_TETRA ) {
        if( isPointInTetra( tn, ntr, p ) ) {
            return ntr;
        } else {
            for( k = 0; k <= 3; ++k ) {
                xtr = tetranet_getSideNext( tn, ntr, k );
                if( isPointInTetra( tn, xtr, p ) ) {
                    return xtr;
                }
            }
        }
    }
    tTetraRef tr;
    for( tr = tn->lastTetraRef; tr != 0; --tr ) {
        if( isPointInTetra( tn, tr, p ) ) {
            return tr;
        }
    }
    return NULL_TETRA;
}
```

```
tTetraRef tetranet_getLastTetraRef( tTetranet tn ) {
    return tn->lastTetraRef;
}

tPointRef tetranet_getLastPointRef( tTetranet tn ) {
    return tn->lastPointRef;
}

unsigned long tetranet_getNumberOfTetras( tTetranet tn ) {
    return tn->numberOfTetras;
}

unsigned long tetranet_getNumberOfPoints( tTetranet tn ) {
    return tn->numberOfPoints;
}

void tetranet_free( tTetranet tn ) {
    atVertex_free( tn );
    nearestp_free( tn );

    free( tn->points );
    free( tn->vertices );
    free( tn->sideArea );
    free( tn->sideNormVect );
    free( tn->sideType );
    free( tn->sideNext );
    free( tn->volume );
    free( tn->states );
    free( tn->massPoint );

    free( tn );
}

void printTetra( tTetranet tn, tTetraRef tr ) {
    printf( "[%7ld] ve: %6ld %6ld %6ld %6ld ",
            tr,
            tetranet_getVertex( tn, tr, 0 ),
            tetranet_getVertex( tn, tr, 1 ),
            tetranet_getVertex( tn, tr, 2 ),
            tetranet_getVertex( tn, tr, 3 ) );
    printf( "nb: %7ld %7ld %7ld %7ld ",
            tetranet_getSideNext( tn, tr, 0 ),
            tetranet_getSideNext( tn, tr, 1 ),
            tetranet_getSideNext( tn, tr, 2 ),
            tetranet_getSideNext( tn, tr, 3 ) );
    printf( "vol: %5.2lf ", tetranet_getTetraVolume( tn, tr ) );
    printf( "sta: %8.4lf ", tetranet_getState( tn, tr, 1 ) );
    printf( "\n" );
}

void printNet( tTetranet tn ) {
    tTetraRef tr;
    for( tr = 1; tr <= tn->lastTetraRef; ++tr ) {
        printTetra( tn, tr );
    }
}
```

testcase.h

```
/*
 *  testcase.h
 *
 *  Tesztesetek, meresekkel
 *  2010-2011 - Martin Jozsef
 */

#ifndef TESTCASE_H_
#define TESTCASE_H_

#include "tetranet.h"

/**
Elinditja a stoppert. Egy teszt kezdeten kell meghivni.
Egyelore csak egy stopper letezik, nem hasznalhato tobb parhuzamosan.
*/
void startClock();

/**
Megallitja a stoppert es kiirja a mert idot, hasznalt memoriat, egyebet.
@param name a mert teszteset neve
*/
void stopClock( char *name );

void test_explode( tTetranet tn );

void test_alfa( tTetranet tn );

void test_nearestp( tTetranet tn );

void test_pointLocation( tTetranet tn );

void test_massPointLocation( tTetranet tn );

void test_delete( tTetranet tn );

void test_flow( tTetranet tn );

#endif
```

testcase.c

```
***
/*
 *  testcase.c
 *
 *  Tesztesetek, meresekkel
 *  2010-2011 - Martin Jozsef
 */

#include "testcase.h"
#include "tetranet.h"
#include "common.h"
#include "nearestp.h"
#include <sys/time.h>
#include <stdio.h>
#include <sys/resource.h>

unsigned long startTime; // ms

void startClock() {
    struct rusage rus;
    getrusage( RUSAGE_SELF, &rus );
    startTime = rus.ru_utime.tv_sec * 1000 + rus.ru_utime.tv_usec / 1000;
}

void stopClock( char *name ) {
    struct rusage rus;
    unsigned long stopTime;
    getrusage( RUSAGE_SELF, &rus );
    stopTime = rus.ru_utime.tv_sec * 1000 + rus.ru_utime.tv_usec / 1000;

    printf( "%3s - %11s (%10.10s..) | ", glob_swName, glob_swDate, glob_inputFile );
    printf( "Test: %8s | ", name );
    printf( "Time: %5.2lf s | ", ( double )( stopTime - startTime ) / 1000.0 );
    printf( "Mem: %8ld kB\n", rus.ru_maxrss );
}

void explode( tTetranet tn, tTetraRef tr ) {
    tPointRef p0, p1, p2, p3, pm;

    p0 = tetranet_getVertex( tn, tr, 0 );
    p1 = tetranet_getVertex( tn, tr, 1 );
    p2 = tetranet_getVertex( tn, tr, 2 );
    p3 = tetranet_getVertex( tn, tr, 3 );
    pm = tetranet_insertPoint( tn, tetranet_getTetraMassPoint( tn, tr ) );

    tetranet_delTetra( tn, tr );

    tetranet_insertTetra( tn, p0, p1, p2, pm );
    tetranet_insertTetra( tn, p0, p1, pm, p3 );
    tetranet_insertTetra( tn, p0, pm, p2, p3 );
    tetranet_insertTetra( tn, pm, p1, p2, p3 );
}

void test_explode( tTetranet tn ) {
    const unsigned count = 50000;
    unsigned i;
    tTetraRef tr;

    startClock();
    tetranet_iteratorInit( tn );
    for( i = 0; i < count; ++i ) {
        tr = tetranet_iteratorNext( tn );
        explode( tn, tr );
    }
    stopClock( "explode" );
}

void test_alfa( tTetranet tn ) {
    const double a = 0.9987;
    const unsigned count = 200;
    double temp = 0;
    tTetraRef tr;
    tTetraRef tr0;
    tTetraRef trMaxVol;
    tSideIndex k;
    unsigned i = 0;

    startClock();
    // nullazas + legnagyobb terfogat keresese
    temp = 0;
    trMaxVol = NULL_TETRA;
    tetranet_iteratorInit( tn );
    while(( tr = tetranet_iteratorNext( tn ) ) != NULL_TETRA ) {
        tetranet_setState( tn, tr, 1, 0.0 );
        if( tetranet_getTetraVolume( tn, tr ) > temp ) {
            trMaxVol = tr;
            temp = tetranet_getTetraVolume( tn, tr );
        }
    }

    // ertek a legnagyobb terfogatuba
    tetranet_setState( tn, trMaxVol, 1, 200.0 );

    for( i = 0; i < count; ++i ) {
        tetranet_iteratorInit( tn );
        // beallitjuk az uj ertekeket states[2]-be
        while(( tr = tetranet_iteratorNext( tn ) ) != NULL_TETRA ) {
            temp = 0;
            for( k = 0; k <= 3; ++k ) {
                tr0 = tetranet_getSideNext( tn, tr, k );
                if( tr0 != NULL_TETRA ) {
                    temp += tetranet_getState( tn, tr0, 1 );
                }
            }
            temp = ( 1 - a ) * temp + a * tetranet_getState( tn, tr, 1 );
            tetranet_setState( tn, tr, 2, temp );
        }
        // visszamasoljuk az ertekeket 2-bol 1-be
        tetranet_iteratorInit( tn );
        while(( tr = tetranet_iteratorNext( tn ) ) != NULL_TETRA ) {
            tetranet_setState( tn, tr, 1, tetranet_getState( tn, tr, 2 ) );
        }
    }
    stopClock( "alfa" );
    printf( "Check value = %lf\n", tetranet_getState( tn, trMaxVol, 1 ) );
}

void test_nearestp( tTetranet tn ) {
    const double epsylon = 0.05;
    unsigned long i;
    tPointRef np;
    tPoint p;
    for( i = 1; i <= tn->lastPointRef; ++i ) {
        p = tetranet_getPoint( tn, i );
        p.x += epsylon;
        p.y += epsylon;
        p.z += epsylon;
        // np = nearestp_findMe(tn,p);
        np = nearestp_search( tn, p );
        if( i != np ) {
            printf( "Nearest to %ld : %ld\n", i, np );
        }
    }
}

void test_massPointLocation( tTetranet tn ) {
    tTetraRef tr;
    tPoint p;
    startClock;
    tetranet_iteratorInit( tn );
    while(( tr = tetranet_iteratorNext( tn ) ) != NULL_TETRA ) {
        if( tetranet_getPointLocation( tn, tetranet_getTetraMassPoint( tn, tr ) ) != tr ) {
            printf( "massPLoc fails by tr=%ld\n", tr );
        }
    }
    stopClock( "massPLoc" );
}

void test_pointLocation( tTetranet tn ) {
    const double c1 = 0.005;
    const double c2 = ( 1 - c1 ) / 3;
    unsigned long i;
    tPointRef np;
    tTetraRef tr;
    tPoint p;
    startClock();

    tetranet_iteratorInit( tn );
    while(( tr = tetranet_iteratorNext( tn ) ) != NULL_TETRA ) {
        tPoint a = tetranet_getPoint( tn, tetranet_getVertex( tn, tr, 0 ) );
        tPoint b = tetranet_getPoint( tn, tetranet_getVertex( tn, tr, 1 ) );
        tPoint c = tetranet_getPoint( tn, tetranet_getVertex( tn, tr, 2 ) );
        tPoint d = tetranet_getPoint( tn, tetranet_getVertex( tn, tr, 3 ) );
        vector v;
        v.x = c2 * ( a.x + b.x + c.x )  + c1 * d.x;
        v.y = c2 * ( a.y + b.y + c.y ) + c1 * d.y ;
        v.z = c2 * ( a.z + b.z + c.z ) + c1 * d.z ;
        if( tetranet_getPointLocation( tn, v ) != tr ) {
            printf( "PointLoc fails by tr=%ld\n", tr );
        }
    }
    stopClock( "PointLoc" );
}

void test_delete( tTetranet tn ) {
    const unsigned long maxCount = 50000;
    unsigned long counter = 0;
    tTetraRef tr = tetranet_getLastTetraRef( tn );
    tTetraRef trn;
    tSideIndex k;
    startClock();
    while( counter < maxCount ) {
        k = 0;
        do {
            trn = tetranet_getSideNext( tn, tr, k );
            ++k;
        } while(( k <= 3 ) && ( trn == NULL_TETRA ) );

//      printf( "c = %ld tr= %ld\n", counter, tr );
        tetranet_delTetra( tn, tr );

        if( trn == NULL_TETRA ) {
            tetranet_iteratorInit( tn );
            trn = tetranet_getLastTetraRef( tn );
        }
        tr = trn;
        ++counter;
    }
    stopClock( "delete" );
}

void test_flow( tTetranet tn ) {
    const unsigned count = 200;
    const double a = 1.0;
    const double dt = 0.0001;

    double temp = 0;
    double uc, un, vc, vn, s;
    tTetraRef tr;
    tTetraRef tr0;
    tTetraRef trMaxVol;
    tSideIndex k;
    unsigned i = 0;

    startClock();

    // homogen feltoltes + legnagyobb terfogat keresese
    temp = 0;
    trMaxVol = NULL_TETRA;
    tetranet_iteratorInit( tn );
    while(( tr = tetranet_iteratorNext( tn ) ) != NULL_TETRA ) {
        tetranet_setState( tn, tr, 0, 0.5 );
        if( tetranet_getTetraVolume( tn, tr ) > temp ) {
            trMaxVol = tr;
            temp = tetranet_getTetraVolume( tn, tr );
        }
    }

    // ertek a legnagyobb terfogatuba
    tetranet_setState( tn, trMaxVol, 0, 0.999 );

    // kezdodik a ciklus
    for( i = 0; i < count; ++i ) {
        tetranet_iteratorInit( tn );
        // beallitjuk az uj ertekeket states[2]-be
        while(( tr = tetranet_iteratorNext( tn ) ) != NULL_TETRA ) {
            temp = 0;
            // sajat allapot
            uc = tetranet_getState( tn, tr, 0 );
            // sajat terfogat
            vn = tetranet_getTetraVolume( tn, tr );
            for( k = 0; k <= 3; ++k ) {
                tr0 = tetranet_getSideNext( tn, tr, k );
                if( tr0 != NULL_TETRA ) {
                    // szomszed allapota
                    un = tetranet_getState( tn, tr0, 0 );
                    // kozos oldal tertulete
                    s  = tetranet_getSideArea( tn, tr, k );
                    // szomszed terfogata
                    vn = tetranet_getTetraVolume( tn, tr0 );
                    // the very secret formula
                    temp += ( -1 * a * ( uc - un ) * s * s ) / ( vc + vn );
                }
            }
            temp = dt * temp + uc;
            tetranet_setState( tn, tr, 1, temp );
        }
        // visszamasoljuk az ertekeket 2-bol 1-be
        tetranet_iteratorInit( tn );
        while(( tr = tetranet_iteratorNext( tn ) ) != NULL_TETRA ) {
            tetranet_setState( tn, tr, 0, tetranet_getState( tn, tr, 1 ) );
        }
    }
    stopClock( "flow" );
    printf( "Check value = %lf\n", tetranet_getState( tn, trMaxVol, 1 ) );
}
```

neighbour.h

```
/*
 *  neighbour.h
 *
 *  Oldalszomszedos tetraederek keresese
 *  2010-2011 - Martin Jozsef
 */

#ifndef NEIGHBOUR_H_
#define NEIGHBOUR_H_

#include "tetranet.h"

void neighbours_update( tTetranet tn );
void neighbours_insert( tTetranet tn, tTetraRef tr );
void neighbours_delete( tTetranet tn, tTetraRef tr );

#endif /* NEIGHBOUR_H_ */
```

neighbour.c

```
/*
 *  neighbour.c
 *
 *  Oldalszomszedos tetraederek keresese
 *  2010-2011 - Martin Jozsef
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "tetranet.h"
#include "errors.h"
#include "atvertex.h"

/// Egy oldalt leiro elem - ezek az elemek kerülnek rendezesre
typedef struct {
    tPointRef  pts[3];    ///< az oldal 3 pontja, index szerint novekvo sorrendben
    unsigned   sideIndex; ///< az oldal indexe a tetraederen belül
    tTetraRef  tetra;     ///< a tetraeder indexe
} element;

element *sarray;

// Osszehasonlito fuggveny def a qsorthoz
typedef int ( *compfn )( const void*, const void* );

// Osszehasonlito fuggveny kifejtese a qsorthoz
int compareElement( element *a, element *b ) {
    if( a->pts[0] > b->pts[0] )
        return 1;
    if( a->pts[0] < b->pts[0] )
        return -1;

    if( a->pts[1] > b->pts[1] )
        return 1;
    if( a->pts[1] < b->pts[1] )
        return -1;
    if( a->pts[2] > b->pts[2] )
        return 1;
    if( a->pts[2] < b->pts[2] )
        return -1;

    return 0;
}

/// Segedfv hibakereseshez
void printarray( long lastIndex ) {
    long i;
    long j;
    for( i = 0; i < lastIndex; i++ ) {
        printf( "%ld: ", ( unsigned long )( sarray[i].tetra ) );
        for( j = 0; j < 3; j++ )
            printf( "%ld ", sarray[i].pts[j] );
        printf( "\n" );
    }
}

/**
 * A tetraederek szomszedossagi viszonyainak kiszamitasa
 * - A teljes halon dolgozik; akkor kell meghivi, ha mar minden tetraeder el van tarolva.
 * - Strategia:
        - Minden oldalhoz letrehozunk egy strukturat (lasd: element), ezekbol tombot alkotunk
        - Ezt a 3 pont indexe szerint sorba rendezzuk
        - Ekkor ha egymas utan ketszer szerepel u.az a pontharmas a tombben,
            akkor a tartalmazo tetraederk szomszedosak.
 * - TODO: Kitol szarmazik ez a modszer?
 */
void neighbours_update( tTetranet tn ) {
    tTetraRef t;
    unsigned long s;
    unsigned long arraySize;
    unsigned long nrOfElements;

    // oldalak tombjenek letrehozasa
    nrOfElements = tetranet_getNumberOfTetras( tn ) * 4;
    arraySize = nrOfElements * sizeof( element );
    sarray = malloc( arraySize );

    // feltoltes
    s = 0;
    tetranet_iteratorInit( tn );
    while(( t = tetranet_iteratorNext( tn ) ) != NULL_TETRA ) {
        sarray[s].pts[0] = tetranet_getVertex( tn, t, 1 );
        sarray[s].pts[1] = tetranet_getVertex( tn, t, 2 );
        sarray[s].pts[2] = tetranet_getVertex( tn, t, 3 );
        sarray[s].tetra = t;
        sarray[s].sideIndex = s % 4;
        ++s;
        sarray[s].pts[0] = tetranet_getVertex( tn, t, 0 );
        sarray[s].pts[1] = tetranet_getVertex( tn, t, 2 );
        sarray[s].pts[2] = tetranet_getVertex( tn, t, 3 );
        sarray[s].tetra = t;
        sarray[s].sideIndex = s % 4;
        ++s;
        sarray[s].pts[0] = tetranet_getVertex( tn, t, 0 );
        sarray[s].pts[1] = tetranet_getVertex( tn, t, 1 );
        sarray[s].pts[2] = tetranet_getVertex( tn, t, 3 );
        sarray[s].tetra = t;
        sarray[s].sideIndex = s % 4;
        ++s;
        sarray[s].pts[0] = tetranet_getVertex( tn, t, 0 );
        sarray[s].pts[1] = tetranet_getVertex( tn, t, 1 );
        sarray[s].pts[2] = tetranet_getVertex( tn, t, 2 );
        sarray[s].tetra = t;
        sarray[s].sideIndex = s % 4;
        ++s;
    }

    // printarray( s );

    // rendezes
    qsort(( void * ) sarray, nrOfElements, sizeof( element ),
          ( compfn ) compareElement );
    // printarray();

    // visszaolvasas, feltoltes
    tTetraRef t0, t1;
    tSideIndex s0, s1;
    s = 0;
    const unsigned ptsSize = sizeof( sarray[0].pts );
    while( s < nrOfElements - 1 ) {
        if( memcmp( sarray[s].pts, sarray[s + 1].pts, ptsSize ) == 0 ) {
            t0 = sarray[s].tetra;
            t1 = sarray[s + 1].tetra;
            s0 = sarray[s].sideIndex;
            s1 = sarray[s + 1].sideIndex;
            if(( tetranet_getSideNext( tn, t0, s0 ) != NULL_TETRA ) ||
                    ( tetranet_getSideNext( tn, t1, s1 ) != NULL_TETRA ) ) {
                exitText( "Inconsistent neighbourhood data." );
            }
            tetranet_setSideNext( tn, t0, s0, t1 );
            tetranet_setSideNext( tn, t1, s1, t0 );
            // ha szomszed, akkor s+1 s+2 mar nem lehet az, atlephetjük a vizsgalatot
            ++s;
        }
        ++s;
    }

// oldalak tombjenek felszabaditasa
    free( sarray );
}

/*  Megkeresi es beallitja adott tetraeder adott oldalahoz tartozo szomszedot.
 *  Beallitja a szomszednal is a kapcsolatot.
 *  Az atVertex-bol dolgozik, feltetel, hogy oda mar helyesen fel legyenek vive a pontok.
 */
void findSideNeighbours( tTetranet tn, tTetraRef tr, tSideIndex si ) {
    tPointRef a, b, c;
    tTetraRef t0 = NULL_TETRA;
    int s0 = -1;    // amig negativ, nincs talalat. amugy a passzolo oldal indexe

    switch( si ) {
    case 0:
        a = tetranet_getVertex( tn, tr, 1 );
        b = tetranet_getVertex( tn, tr, 2 );
        c = tetranet_getVertex( tn, tr, 3 );
        break;
    case 1:
        a = tetranet_getVertex( tn, tr, 0 );
        b = tetranet_getVertex( tn, tr, 2 );
        c = tetranet_getVertex( tn, tr, 3 );
        break;
    case 2:
        a = tetranet_getVertex( tn, tr, 0 );
        b = tetranet_getVertex( tn, tr, 1 );
        c = tetranet_getVertex( tn, tr, 3 );
        break;
    case 3:
        a = tetranet_getVertex( tn, tr, 0 );
        b = tetranet_getVertex( tn, tr, 1 );
        c = tetranet_getVertex( tn, tr, 2 );
        break;
    default:
        a = b = c = NULL_POINT;
        exitText( "Index failed by findNeighbours" );
    }
    // az a ponthoz keresünk talalatot
    atVertex_init( tn, a );
    while(( s0 < 0 ) && (( t0 = atVertex_next( tn ) ) != NULL_TETRA ) ) {
        if( t0 != tr ) {
            if( a == tetranet_getVertex( tn, t0, 0 ) ) {
                if( b == tetranet_getVertex( tn, t0, 1 ) ) {
                    if( c == tetranet_getVertex( tn, t0, 2 ) ) {
                        s0 = 3;
                    } else if( c == tetranet_getVertex( tn, t0, 3 ) ) {
                        s0 = 2;
                    }
                } else if( b == tetranet_getVertex( tn, t0, 2 ) ) {
                    if( c == tetranet_getVertex( tn, t0, 3 ) ) {
                        s0 = 1;
                    }
                }
            } else if( a == tetranet_getVertex( tn, t0, 1 ) ) {
                if( b == tetranet_getVertex( tn, t0, 2 ) ) {
                    if( c == tetranet_getVertex( tn, t0, 3 ) ) {
                        s0 = 0;
                    }
                }
            }
        }
    }

    if( s0 < 0 ) {
        tetranet_setSideNext( tn, tr, si, NULL_TETRA );
    } else {
        if(( tetranet_getSideNext( tn, t0, s0 ) != NULL_TETRA ) ||
                ( tetranet_getSideNext( tn, tr, si ) != NULL_TETRA ) ) {
            exitText( "Inconsistent neighbourhood data." );
        }
        tetranet_setSideNext( tn, tr, si, t0 );
        tetranet_setSideNext( tn, t0, s0, tr );
    }
}

void neighbours_insert( tTetranet tn, tTetraRef tr ) {
    int k;
    for( k = 0; k <= 3; k++ ) {
        findSideNeighbours( tn, tr, k );
    }
}

void neighbours_delete( tTetranet tn, tTetraRef tr ) {
    tSideIndex k;
    tSideIndex j;
    tTetraRef nb;
    for( k = 0; k <= 3; k++ ) {
        if(( nb = tetranet_getSideNext( tn, tr, k ) ) != NULL_TETRA ) {
            for( j = 0; tetranet_getSideNext( tn, nb, j ) != tr; ++j ) {
                if( j > 3 )
                    exitText( "neighbours_delete error: asymmetric neighbourhood." );
            }
            tetranet_setSideNext( tn, nb, j, NULL_TETRA );
            tetranet_setSideNext( tn, tr, k, NULL_TETRA );
        }
    }
}
```

nearestp.h

```
/*
 *  nearestp.h
 *
 *  Kd-tree es nearest neighbour search
 *  2010-2011 - Martin Jozsef
 */

#ifndef NEARESTP_H_
#define NEARESTP_H_

#include "tetranet.h"

/**
 *  A keresest segito kd-tree elokeszitese
 */
void nearestp_update( tTetranet tn );

/**
 *  A legkozelebbi pont megkeresese
 *  @param p a kerdeses uj pont (koordinatakkal megadva)
 *  @return a ponthalmaz legkozelebbi pontjanak indexe
 */
tPointRef nearestp_search( tTetranet tn, tPoint p );

/**
 *  Uj pont beszurasa a keresofaba
 *  @param p a beszurando pont indexe
 */
void nearestp_addPoint( tTetranet tn, tPointRef p );

/**
 *  A kd-tree altalt hasznalt memoria felszabaditasa
 */
void nearestp_free( tTetranet tn );

#endif /* NEARESTP_H_ */
```

nearestp.c

```
/*
 *  nearestp.c
 *
 *  Kd-tree es nearest neighbour search
 *  2010-2011 - Martin Jozsef
 */

#include "nearestp.h"
#include "tetranet.h"
#include "errors.h"
#include <stdio.h>
#include <stdlib.h>

typedef struct sctNode {
    tPointRef value;
    struct sctNode *left;
    struct sctNode *right;
} node;

typedef struct {
    tPointRef idx;
    double x;
    double y;
    double z;
} tmpElement;

tmpElement *tmpArray;

void printNode( node *n, int depth ) {
    if( n != NULL ) {
        if( n->left != NULL ) printf( "%d: %ld -> %ld\n", depth, n->value, n->left->value );
        if( n->right != NULL ) printf( "%d: %ld -> %ld\n", depth, n->value, n->right->value );
        if(( n->left == NULL ) && ( n->right == NULL ) ) printf( "%ld egy level.\n", n-
>value );
        ++depth;
        printNode( n->left, depth );
        printNode( n->right, depth );
    }
}

void printTree( tTetranet tn ) {
    printf( "\ndigraph G{" );
    printNode(( node* )( tn->nearestp ), 0 );
    printf( "}\n" );
}

// Osszehasonlito fuggveny def a qsorthoz
typedef int ( *compfn )( const void*, const void* );

// Osszehasonlito fuggveny kifejtesei a qsorthoz
int compareByX( const tmpElement *a, const tmpElement *b ) {
    double temp = ( a->x ) - ( b->x );
    if( temp > 0.0 ) return 1;
    else if( temp < 0.0 ) return -1;
    else return 0;
}

int compareByY( const tmpElement *a, const tmpElement *b ) {
    double temp = ( a->y ) - ( b->y );
    if( temp > 0.0 ) return 1;
    else if( temp < 0.0 ) return -1;
    else return 0;
}

int compareByZ( const tmpElement *a, const tmpElement *b ) {
    double temp = ( a->z ) - ( b->z );
    if( temp > 0.0 ) return 1;
    else if( temp < 0.0 ) return -1;
    else return 0;
}

node *buildKdTree( tPointRef first, tPointRef last, unsigned int depth ) {
    node *tmpNode;
    tPointRef median;

    if( last < first ) {
        return NULL;
    } else {
        // sort by axis
        switch( depth % 3 ) {
        case 0:
            qsort(( void * ) &tmpArray[first], 1 + last - first, sizeof( tmpElement ),
( compfn ) compareByX );
            break;
        case 1:
            qsort(( void * ) &tmpArray[first], 1 + last - first, sizeof( tmpElement ),
( compfn ) compareByY );
            break;
        case 2:
            qsort(( void * ) &tmpArray[first], 1 + last - first, sizeof( tmpElement ),
( compfn ) compareByZ );
            break;
        }
        // Sort point list and choose median as pivot element
        median = ( first + last ) / 2;
        tmpNode = malloc( sizeof( node ) );
        tmpNode->value = tmpArray[median].idx;
        tmpNode->left = buildKdTree( first, median - 1, depth + 1 );
        tmpNode->right = buildKdTree( median + 1, last, depth + 1 );

        return tmpNode;
    }
}

void nearestp_update( tTetranet tn ) {
    tPointRef i = 1;
    tPoint p;
    unsigned long nbp = tetranet_getNumberOfPoints( tn );
    tmpArray = malloc(( nbp + 1 ) * sizeof( tmpElement ) );
    for( i = nbp; i != 0; --i ) {
        p = tetranet_getPoint( tn, i );
        tmpArray[i].idx = i;
        tmpArray[i].x = p.x;
        tmpArray[i].y = p.y;
        tmpArray[i].z = p.z;
    }
    tn->nearestp = buildKdTree( 1, nbp, 0 );
    free( tmpArray );
//   printTree( tn );
}

inline double distance( tPoint a, tPoint b ) {
    return ( b.x - a.x ) * ( b.x - a.x ) +
           ( b.y - a.y ) * ( b.y - a.y ) +
           ( b.z - a.z ) * ( b.z - a.z );
}

tPointRef kdsearch( tTetranet tn, node *here, tPoint point, tPointRef best, unsigned int depth
) {
    if( here == NULL ) {
        return best;
    }

    if( best == NULL_POINT ) {
        best = here->value;
    }
    tPoint phere = tetranet_getPoint( tn, here->value );
    tPoint pbest = tetranet_getPoint( tn, best );
    if( distance( phere, point ) < distance( pbest, point ) ) {
        best = here->value;
        pbest = tetranet_getPoint( tn, best );
    }

    double d;
    switch( depth % 3 ) {
    case 0:
        d = point.x - phere.x;
        break;
    case 1:
        d = point.y - phere.y;
        break;
    case 2:
        d = point.z - phere.z;
        break;
    default:
        d = 0.0;
        exitText( "Switch failure in nearest.c." );
    }

    if( d < 0 ) {
        best = kdsearch( tn, here->left, point, best, depth + 1 );
        pbest = tetranet_getPoint( tn, best );
        if(( d * d ) < distance( pbest, point ) ) {
            best =  kdsearch( tn, here->right, point, best, depth + 1 );
        }
    } else {
        best = kdsearch( tn, here->right, point, best, depth + 1 );
        pbest = tetranet_getPoint( tn, best );
        if(( d * d ) < distance( pbest, point ) ) {
            best =  kdsearch( tn, here->left, point, best, depth + 1 );
        }
    }
    return best;
}

tPointRef nearestp_search( tTetranet tn, tPoint p ) {
    return kdsearch( tn, tn->nearestp, p, NULL_POINT , 0 );
}

/* TODO: A beszurasokkal a fa lassan elveszti kiegyensulyozott jelleget,
 * ezert egy idö utan ujra kellene rendezni.
 */
void nearestp_addPoint( tTetranet tn, tPointRef pr ) {
    unsigned int depth = 0;
    node *parent = tn->nearestp;
    node *newNode =  malloc( sizeof( node ) );
    bool found = FALSE;
    tPoint p = tetranet_getPoint( tn, pr );
    tPoint h;
    double d;

    newNode->left = NULL;
    newNode->right = NULL;
    newNode->value = pr;

    while( !found ) {
        h = tetranet_getPoint( tn, parent->value );
        switch( depth % 3 ) {
        case 0:
            d = p.x - h.x;
            break;
        case 1:
            d = p.y - h.y;
            break;
        case 2:
            d = p.z - h.z;
            break;
        default:
            d = 0.0;
            exitText( "Switch failure in nearest.c." );
        }
        if( d < 0 ) {
            if( parent->left == NULL ) {
                parent->left = newNode;
                found = TRUE;
            } else {
                parent = parent->left;
            }
        } else {
            if( parent->right == NULL ) {
                parent->right = newNode;
                found = TRUE;
            } else {
                parent = parent->right;
            }
        }
        ++depth;
    }
}

void freeNode( node *n ) {
    if( n != NULL ) {
        freeNode( n->left );
        freeNode( n->right );
        free( n );
    }
}

void nearestp_free( tTetranet tn ) {
    freeNode( tn->nearestp );
    tn->nearestp = NULL;
}
```

nasreader.h

```
/*
 *  nasreader.h
 *
 *  nastran fajlok olvasasa, az adatok konvertalasa es elemkenti tovabbadasa
 *  2010-2011 - Martin Jozsef
 */

#ifndef NASREADER_H_
#define NASREADER_H_

#include "tetranet.h"
#include "common.h"
#include <stdio.h>

unsigned long nasreader_getPointNr( FILE *f );
unsigned long nasreader_getTetraNr( FILE *f );
void nasreader_readFirstPoint( FILE *f, tPoint *p );
bool nasreader_readNextPoint( FILE *f, tPoint *p );
void nasreader_readFirstTetra( FILE *f, tPointRef *p );
bool nasreader_readNextTetra( FILE *f, tPointRef *p );

#endif /* NASREADER_H_ */
```

nasreader.c

```
/*
 *  nasreader.c
 *
 *  nastran fajlok olvasasa, az adatok konvertalasa es elemkenti tovabbadasa
 *  2010-2011 - Martin Jozsef
 */

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>

#include "nasreader.h"
#include "common.h"
#include "errors.h"
#include "tetranet.h"

#define MAX_LINE 99

char line[MAX_LINE + 1];
char strTemp[9]   = "01234567\0";
char strPoint[10] = "012345678\0";

/*
  csak mert a NAS a 3.17E-3 helyett a 3.17-3 format szereti. Juhuuuu.
*/
void expHack( char *str ) {
#define SPC ' '
#define MINUS '-'
    unsigned short i = 8;
    unsigned short k = 9;

    do {
        --i;
        --k;
        str[k] = str[i];
        if(( str[i] == MINUS)  && ( i > 0 ) && ( str[i-1] != SPC )) {
        --k;
        str[k] = 'E';
        }
    } while( i != 0 );
    if( k > 1 ) {
        exitText( "point format error" );
    }
    if( k == 1 ) {
        str[0] = SPC;
    }
}

inline void line2point( tPoint *p ) {
    // a stringet csak a deklaracioban zartuk le,
    // ezert a terminatort nem szabad felulirni
    strncpy( strPoint, &line[24], 8 );
    expHack( strPoint );
    p->x = atof( strPoint );
    strncpy( strPoint, &line[32], 8 );
    expHack( strPoint );
    p->y = atof( strPoint );
    strncpy( strPoint, &line[40], 8 );
    expHack( strPoint );
    p->z = atof( strPoint );
    // TODO mi van, ha az atof hibaval ter vissza?
}

inline void line2tetra( tPointRef *p ) {
    // a stringet csak a deklaracioban zartuk le,
    // ezert a terminatort nem szabad felulirni
    strncpy( strTemp, &line[24], 8 );
    p[0] = atoi( strTemp );
    strncpy( strTemp, &line[32], 8 );
    p[1] = atoi( strTemp );
    strncpy( strTemp, &line[40], 8 );
    p[2] = atoi( strTemp );
    strncpy( strTemp, &line[48], 8 );
    p[3] = atoi( strTemp );
    // TODO atoi hibajelzesek figyelese
}

unsigned long int nasreader_getPointNr( FILE *f ) {
    unsigned long int i = 0;
    rewind( f );
    do {
        if( !fgets( line, MAX_LINE, f ) ) {
            exitText( "No GRID line found in the input file." );
        }
    } while( strncmp( line, "GRID", 4 ) != 0 );
    do {
        ++i;
        if( !fgets( line, MAX_LINE, f ) ) {
            exitText( "Suddenly EOF by reading Grids." );
        }
    }  while( strncmp( line, "GRID", 4 ) == 0 );
    return i;
}

unsigned long int nasreader_getTetraNr( FILE *f ) {
    unsigned long int i = 0;
    rewind( f );
    do {
        if( !fgets( line, MAX_LINE, f ) ) {
            exitText( "No CTETRA line found in the input file." );
        }
    } while( strncmp( line, "CTETRA", 6 ) != 0 );
    do {
        ++i;
        if( !fgets( line, MAX_LINE, f ) ) {
            exitText( "Suddenly EOF by reading Grids." );
        }
    } while( strncmp( line, "CTETRA", 6 ) == 0 );
    return i;
}

void nasreader_readFirstPoint( FILE *f, tPoint *p ) {
    rewind( f );
    while( TRUE ) {
        if( !fgets( line, MAX_LINE, f ) ) {
            exitText( "No GRID line found in the input file." );
        }
        if( strncmp( line, "GRID", 4 ) == 0 ) {
            break;
        }
    }
    line2point( p );
}

/*
 * 0       8       16      24      32      40
 * |       |       |       |       |       |       |
 * GRID          27        93.9291 -23.035893.43689
 *
 */
bool nasreader_readNextPoint( FILE *f, tPoint *p ) {
    // a kovetkezo GRID sor keresese
    if( !fgets( line, MAX_LINE, f ) ) {
        exitText( "Suddenly end of file during reading grids." );
    }
    if( strncmp( line, "GRID", 4 ) != 0 ) {
        return FALSE;
    }
    line2point( p );
    return TRUE;
}

void nasreader_readFirstTetra( FILE *f, tPointRef *p ) {
    rewind( f );
    while( TRUE ) {
        if( !fgets( line, MAX_LINE, f ) ) {
            exitText( "No CTETRA line found in the input file." );
        }
        if( strncmp( line, "CTETRA", 6 ) == 0 ) {
            break;
        }
    }
    line2tetra( p );
}

/*
 * 0       8       16      24      32      40      48
 * |       |       |       |       |       |       |
 * CTETRA         1       8   21185   20773     380   20488
 *
 */
bool nasreader_readNextTetra( FILE *f, tPointRef *p ) {
    // a kovetkezo CTETRA sor keresese
    if( !fgets( line, MAX_LINE, f ) ) {
        exitText( "Suddenly end of file during reading grids." );
    }
    if( strncmp( line, "CTETRA", 6 ) != 0 ) {
        return FALSE;
    }
    line2tetra( p );
    return TRUE;
}
```

atvertex.h

```
/*
 *  atvertex.h
 *
 *  Adott ponthoz tartozo tetraederek keresese.
 *  2010-2011 - Martin Jozsef
 */

#ifndef ATVERTEX_H_
#define ATVERTEX_H_

#include "tetranet.h"
#include "common.h"

/**
 *  Letrehozza a keresest gyorsito adatstrukturat. (magyaran indexel)
 */
void atVertex_update( tTetranet tn );

/**
 *  Elokeszuleti lepes adott ponthoz valo kereseshez.
 *  Eloszor mindig ezt kell meghivni, csak utana lehet a atVertex_getNext-et
 *  @param p a kerdeses pont indexe
 *  @return TRUE, ha van a ponthoz tetraeder tarolva.
 */
bool atVertex_init( tTetranet tn, tPointRef p );

/**
 *  Az adott ponthoz tartozo kovetkezo tatraeder indexet adja vissza.
 *  Parametere nincs, a atVertex_getFirst-ben megadott ponthoz keres.
 *  @return a ponthoz tartozo kovetkezo tetraeder indexe, ha nincs tobb, akkor NULL_TETRA
 */
tTetraRef atVertex_next( tTetranet tn );

/**
 *  A lefoglalt területek felszabaditasa
 */
void atVertex_free( tTetranet tn );

/**
 *  Uj tetraeder indexelese
 *  A rutin johiszemuen feltetelezi, hogy a tetraeder meg nem szerepel.
 */
void atVertex_insert( tTetranet tn, tTetraRef tr );

/**
 *  Tetraeder eltavolitasa az atvertex nyilvantartasbol
 *  A rutin johiszemuen feltetelezi, hogy a tetraeder korabban fel lett veve.
 */
void atVertex_delete( tTetranet tn, tTetraRef tr );
#endif /* ATVERTEX_H_ */
```

atvertex.c

```
/*
 *  atvertex.c
 *
 *  Adott ponthoz tartozo tetraederek keresese. hash-tablas verzio
 *  2010-2011 - Martin Jozsef
 */

#include <stdlib.h>
#include <string.h>
#include "atvertex.h"
#include "tetranet.h"
#include "common.h"
#include "errors.h"

struct tElementX {
    tTetraRef tRef;
    struct tElementX *next;
```

```c
};
typedef struct tElementX tElement;

typedef struct {
    tElement **idxArr; // pointerek dinamikus tombje
    tPointRef maxPr;
    tElement *act;
} tAtVertexDesc;

/**
 *  Letrehozza a keresest gyorsito adatstrukturat. (magyaran indexel)
 */
void atVertex_update( tTetranet tn ) {
    unsigned long len;
    tTetraRef tr;
    tAtVertexDesc *atv;
    tElement *dummyPointer;

    // elokeszites
    atVertex_free( tn );
    atv = malloc( sizeof( tAtVertexDesc ) );
    /* csak mert nem akarok referenciat szamossagkent hasznalni, kiszedtem:
    atv->maxPr = tetranet_getLastPointRef( tn );
    */
    atv->maxPr = tetranet_getNumberOfPoints( tn );
    len = ( atv->maxPr + 1 ) * sizeof( dummyPointer );
    atv->idxArr = malloc( len );
    memset( atv->idxArr, '\0', len );
    atv->act = NULL;
    tn->atVertex = atv;
    // feltoltes
    tetranet_iteratorInit( tn );
    while(( tr = tetranet_iteratorNext( tn ) ) != NULL_TETRA ) {
        atVertex_insert( tn, tr );
    }
}

/**
 *  Elokeszuleti lepes adott ponthoz valo kereseshez.
 *  Eloszor mindig ezt kell meghivni,
 *  csak utana lehet az atVertex_getNext-et
 *  @param p a kerdeses pont indexe
 *  @return FALSE, ha a pont nincs indexelve
 */
bool atVertex_init( tTetranet tn, tPointRef p ) {
    tAtVertexDesc *atv = tn->atVertex;
    if( p > atv->maxPr ) {
        atv->act = NULL;
        return FALSE;
    } else {
        atv->act = atv->idxArr[p];
        return TRUE;
    }
}

/**
 *  Az adott ponthoz tartozo kovetkezo tetraeder indexet adja vissza.
 *  Csak az init utan van ertelme meghivni,
 *  ott kell megadni, hogy melyik ponthoz keresunk.
 *  @return a ponthoz tartozo kovetkezo tetraeder indexe, ha nincs tobb akkor NULL_TETRA
 */
tTetraRef atVertex_next( tTetranet tn ) {
    tAtVertexDesc *atv = tn->atVertex;
    if( atv->act == NULL ) {
        return NULL_TETRA;
    } else {
        tTetraRef tr = atv->act->tRef;
        atv->act = atv->act->next;
        return tr;
    }
}

void atVertex_free( tTetranet tn ) {
    tAtVertexDesc *atv = tn->atVertex;
    tElement *tmp;
    if( atv != NULL ) {
        unsigned long i = 0;
        for( i = 0; i <= atv->maxPr; ++i ) {
            while( atv->idxArr[i] != NULL ) {
                tmp = atv->idxArr[i]->next;
                free( atv->idxArr[i] );
                atv->idxArr[i] = tmp;
            }
        }
        free( atv->idxArr );
        free( tn->atVertex );
        tn->atVertex = NULL;
    }
}

void atVertex_insert( tTetranet tn, tTetraRef tr ) {
    tAtVertexDesc *atv = tn->atVertex;
    unsigned i;
    tElement *elem;
    tPointRef pr, k;

    for( i = 0; i <= 3; i++ ) {
        pr = tetranet_getVertex( tn, tr, i );
        if( pr > atv->maxPr ) {
            // boviteni kell az indextombot. legyen 110%-os az uj meret
            atv->idxArr = realloc( atv->idxArr, ( pr + pr / 10 + 1 ) * sizeof( elem ) );
            for( k = atv->maxPr + 1; k <= pr; ++k ) {
                atv->idxArr[k] = NULL;
            }
            atv->maxPr = pr;
        }
        elem = malloc( sizeof( tElement ) );
        elem->next = atv->idxArr[pr];
        elem->tRef = tr;
        atv->idxArr[pr] = elem;
    }
}

void atVertex_delete( tTetranet tn, tTetraRef tr ) {
    int k;
    tElement *elem;
    tElement *prev;
    tPointRef pr;
    tAtVertexDesc *atv = tn->atVertex;
    bool done = FALSE;

    for( k = 0; k <= 3; k++ ) {
        pr = tetranet_getVertex( tn, tr, k );
        elem = atv->idxArr[pr];
        prev = NULL;
        done = FALSE;
        while( !done ) {
            if( elem == NULL ) {
                exitText( "atVertex_delete error: vertex isn't indexed." );
            }
            if( elem->tRef == tr ) {
                if( prev == NULL ) {
                    atv->idxArr[pr] = elem->next;
                } else {
                    prev->next = elem->next;
                }
                free( elem );
                done = TRUE;
            }
            prev = elem;
            elem = elem->next;
        }
    }
}
```

vector.h

```
/*
 *  vector.h
 *
 *  Vektorok, vektormuveletek
 *  2010-2011 - Martin Jozsef
 */

#ifndef VECTOR_H_
#define VECTOR_H_

/// vektor, harom dimenzios koordinatakkal
typedef struct {
    double x;
    double y;
    double z;
} vector;

/// vektor iranyanak megforditasa
vector negativeVector( vector a );

/// ket vektor osszege
vector vector_add( vector a, vector b );

/// a - b vektor, azaz b-bol a-ba mutato vektor
vector vector_diff( vector a, vector b );

/// vektor konstansszorosa
vector vector_constMult( vector a, double c );

/// vektor hossza
double vector_length( vector a );

/// skalar szorzat
double dotProduct( vector a, vector b );

/// vektorialis szorzat
vector crossProduct( vector a, vector b );

/// vegyes szorzat
double tripleProduct( vector a, vector b, vector c );

/// harom ponttal megadott sik normalvektora
vector normalOfPlane( vector a, vector b, vector c );

/// tetraeder tomegkozeppontja
vector massPoint( vector a, vector b, vector c, vector d );

#endif
```

vector.c

```
/*
 *  vector.c
 *
 *  Vektorok, vektormuveletek
 *  2010-2011 - Martin Jozsef
 */

#include "vector.h"
#include <math.h>

vector vector_add( vector a, vector b ) {
    vector v;
    v.x = a.x + b.x;
    v.y = a.y + b.y;
    v.z = a.z + b.z;
    return v;
}
```

```
vector vector_diff( vector a, vector b ) {
    vector v;
    v.x = a.x - b.x;
    v.y = a.y - b.y;
    v.z = a.z - b.z;
    return v;
}

vector vector_constMult( vector a, double c ) {
    vector v;
    v.x = c * a.x;
    v.y = c * a.y;
    v.z = c * a.z;
    return v;
}

double vector_length( vector a ) {
    return sqrt( a.x * a.x + a.y * a.y + a.z * a.z );
}

vector negativeVector( vector a ) {
    a.x = -a.x;
    a.y = -a.y;
    a.z = -a.z;
    return a;
}

double dotProduct( vector a, vector b ) {
    return a.x * b.x + a.y * b.y + a.z * b.z;
}

vector crossProduct( vector a, vector b ) {
    vector n;
    n.x = a.y * b.z - a.z * b.y;
    n.y = a.z * b.x - a.x * b.z;
    n.z = a.x * b.y - a.y * b.x;
    return n;
}

double tripleProduct( vector a, vector b, vector c ) {
    return dotProduct( a, crossProduct( b, c ) );
}

vector normalOfPlane( vector a, vector b, vector c ) {
    return crossProduct( vector_diff( a, b ), vector_diff( a, c ) );
}

vector massPoint( vector a, vector b, vector c, vector d ) {
    vector v;
    v.x = ( a.x + b.x + c.x + d.x ) / 4;
    v.y = ( a.y + b.y + c.y + d.y ) / 4;
    v.z = ( a.z + b.z + c.z + d.z ) / 4;
    return v;
}
```

errors.h

```
/*
 *  errors.h
 *
 *  Hibajelzesek kiirasa
 *  2010-2011 - Martin Jozsef
 */

#ifndef ERRORS_H_
#define ERRORS_H_

#define MAX_ERROR_STRING 99

void errorText( char *text );
void exitText( char *text );
void debugText( char *text );

#endif
```

errors.c

```
/*
 *  errors.c
 *
 *  Hibajelzesek kiirasa
 *  2010-2011 - Martin Jozsef
 */

#include "errors.h"
#include <stdio.h>
#include <stdlib.h>

void errorText( char *text ) {
    printf( "ERROR: %s\n", text );
}

void exitText( char *text ) {
    errorText( text );
    exit( EXIT_FAILURE );
}

void debugText( char *text ) {
    printf( "DEBUG: %s\n", text );
}
```

common.h

```
/*
 *  common.h
 *
 *  Altalanos informaciok, konstansok
 *  2010-2011 - Martin Jozsef
 */

#ifndef COMMON_H_
#define COMMON_H_

/// boolean tipus definialasa
typedef int bool;
#define FALSE 0
#define TRUE (!FALSE)

/// nagyon kicsi szam, long tipusu nulla
#define EPS 0.0000000001

/// a programvaltozat esetleg verzio jelzesere, ertekadas a main.c-ben, max 8 karakter
char glob_swName[9];
/// a parameterkent megadott fajlnev
char glob_inputFile[64];
/// a buildeles datuma, a preprocesszor szerinti formatumban
char glob_swDate[12];

void common_setGlobSwName( const char *in );
void common_setGlobSwDate( );
void common_setGlobInputFile( const char *in );

#endif
```

common.c

```
/*
 *  common.c
 *
 *  Altalanos informaciok, konstansok
 *  2010-2011 - Martin Jozsef
 */

#include <string.h>
#include "common.h"

#define DELIMITER '/'

void common_setGlobSwName( const char *in ) {
    int i = 0;
    int k = 0;
    int l = strlen( in );
    for( i = 0; i < l; i++ )
        if( in[i] == DELIMITER )
            k = i + 1;
    strncpy( glob_swName, &in[k], l - k );
}

void common_setGlobSwDate( ) {
    strncpy( glob_swDate, __DATE__, 11 );
}

void common_setGlobInputFile( const char *in ) {
    int i = 0;
    int k = 0;
    int l = strlen( in );
    for( i = 0; i < l; i++ )
        if( in[i] == DELIMITER )
            k = i + 1;
    strncpy( glob_inputFile, &in[k], l - k );
}
```

A CD melléklet tartalma

```
.
├── dokumentumok
│   ├── bemutato
│   │   ├── bemutato         - a bemutató HTML formátumban (mappa, 20 html és 10 png fájl)
│   │   ├── bemutato.odp     - a bemutató szerkeszthető odp formátumban
│   │   └── bemutato.pdf     - a bemutató pdf formátumban
│   ├── dolgozat
│   │   ├── diplomamunka.odt - a teljes diplomadolgozat - szerkeszthető
│   │   └── diplomamunka.pdf - a teljes diplomadolgozat - pdf
│   ├── kivonat
│   │   ├── kivonat.odt      - a kivonat - szerkeszthető
│   │   └── kivonat.pdf      - a kivonat - pdf
│   ├── melleklet
│   │   ├── melleklet.odt    - a melléklet - szerkeszthető
│   │   └── melleklet.pdf    - a melléklet - pdf
│   └── summary
│       ├── summary_en.html  - összefoglalás - angol nyelven
│       └── summary_hu.html  - összefoglalás - magyar nyelven
├── forraskod
│   ├── base                 - a közösen hasznalt forrasfajlok
│   │   ├── atvertex.c
│   │   ├── atvertex.h
│   │   ├── common.c
│   │   ├── common.h
│   │   ├── errors.c
│   │   ├── errors.h
│   │   ├── main.c
│   │   ├── nasreader.c
│   │   ├── nasreader.h
│   │   ├── nearestp.c
│   │   ├── nearestp.h
│   │   ├── neighbour.c
│   │   ├── neighbour.h
│   │   ├── testcase.c
│   │   ├── testcase.h
│   │   ├── vector.c
│   │   └── vector.h
│   ├── test.sh               - a tesztet indító szkript
│   ├── v3                    - a v3 változat saját tetraéder modulja és project fájlja
│   │   ├── tetranet.c
│   │   ├── tetranet.h
│   │   └── v3.cbp
│   ├── v3x                   - a v3x változat saját tetraéder modulja és project fájlja
│   │   ├── tetranet.c
│   │   ├── tetranet.h
│   │   └── v3x.cbp
│   ├── v3y                   - a v3y változat saját tetraéder modulja és project fájlja
│   │   ├── tetranet.c
│   │   ├── tetranet.h
│   │   └── v3y.cbp
│   ├── v4                    - a v4 változat saját tetraéder modulja és project fájlja
│   │   ├── tetranet.c
│   │   ├── tetranet.h
│   │   └── v4.cbp
│   ├── v5                    - a v5 változat saját tetraéder modulja és project fájlja
│   │   ├── tetranet.c
│   │   ├── tetranet.h
│   │   └── v5.cbp
│   ├── v5x                   - a v5x változat saját tetraéder modulja és project fájlja
│   │   ├── tetranet.c
│   │   ├── tetranet.h
│   │   └── v5x.cbp
│   └── v5y                   - a v5y változat saját tetraéder modulja és project fájlja
│       ├── tetranet.c
│       ├── tetranet.h
│       └── v5y.cbp
├── tartalom.txt              - a cd melléklet tartalomjegyzéke
└── teszt
    ├── input
    │   ├── fuvoka_640000_mod.bdf        - a teszteleshez használt nagyobbik hálózat leíró fájlja
    │   └── szivocso_vol_tetra_hm.nas    - a teszteleshez használt kisebbik hálózat leíró fájlja
    └── output
        └── test.txt                     - a teljes teszt kimenete
```