```
/// @file QuadTree.hpp
/// @mainpage NHF 2013 - QuadTree (Négy elágazású duplán láncolt generikus fa)
/// Feladat kiírás:
/// Készítsen GENERIKUS duplán láncolt 4 elágazású fát (quad-tree)!
/// Valósítsa meg az összes értelmes műveletet operátor átdefiniálással
    (overload),
/// de nem kell ragaszkodni az összes operátor átdefiniálásához!
/// Amennyiben lehetséges használjon iterátort!
///  @author Tóth András (O8POUA)

template <class T>
class Point;

template <class T>
class QuadTree;

template <class T>
class QuadTreeNode;

template <class T>
std::ostream& operator<<(std::ostream &,const Point<T> &);

template <class T>
std::ostream& operator<<(std::ostream &,const QuadTree<T> &);

template <class T>
std::ostream& operator<<(std::ostream &,const QuadTreeNode<T> &);

/// @brief Point (pont) osztály. A négy elágazású generikus fát ilyen
    pontokkal tölthetjük fel.
template <class T>
class Point{
    /// @brief Az adat helye.
    double x, y;
    /// @brief Az adat.
    T data;

public:
    /// @brief Default konstruktor.
    Point(): x(0), y(0) {}

    /// @brief Konstruktor adattal.
    Point(T data, double x=0, double y=0): data(data), x(x), y(y) {}

    /// @brief operator==, két pont egyenlősége vizsgálható. Igaz, ha a két
        pont minden adata megegyezik.
    bool operator==(Point & point){
        return (x==point.x && y==point.y && data==point.data);
    }

    /// @brief operator!=, két pont egyenlősége vizsgálható. Igaz, ha a két
        pont valamelyik adata nem egyezik meg.
    bool operator!=(Point & point){
        return !((*this) == point);
    }

    T& getData(){return data;}

    friend class QuadTree<T>;
```

```
    friend class QuadTreeNode<T>;
    friend std::ostream& operator<< <T>(std::ostream &, const Point &);
};

/// @brief QuadTreeNode (négy elágazású generikus fa csomópontja) osztály. A
    négy elágazású generikus fa ilyen csomópontokból épül fel.
template <class T>
class QuadTreeNode{
    /// @brief Szülő.
    QuadTreeNode *parent;

    /// @brief Négy gyerek.
    QuadTreeNode *children[4];

    /// @brief Pont, melyben az adatot tárolhatjuk.
    Point<T> *point;

    /// @brief Pontok száma.
    size_t number_of_points;

    /// @brief A területre jellemző adatok. (x ; y) pont a téglalap bal alsó
        pontja.
    double x, y, width, height;

    /// @brief Melyik szinten található a fában? A gyökér van az 1. szinten.
    unsigned level;

    /// @brief Milyen mélységig nőhet a fa? Ezzel a statikus változóval lehet
        beállítani.
    static unsigned MAX_LEVEL;

    /// @brief Nincs paraméter nélküli konstruktora.
    QuadTreeNode();

    /// @brief QuadTree és QuadTreeNode hozhat csak létre új csomópontot.
        (Adat nélkül)
    QuadTreeNode(double width, double height, double x=0, double y=0, unsigned
        level=1, QuadTreeNode *parent=NULL): parent(parent), children{NULL,
        NULL, NULL, NULL}, width(width), height(height), x(x), y(y), level
        (level), point(NULL), number_of_points(0){}

    /// @return A csomópont rendelkezik-e adattal?
    bool hasData() const{
        return number_of_points==0 ? false : true;
    }

    /// @brief Terület felbontása négy egybevágó téglalapra.
    void split();

    /// @brief Új pont / adat beszúrása.
    void insert(Point<T> new_point);

    /// @brief Csomópontról eldönti hogy levél-e.
    /// @return Levél?
    bool isLeaf() const{
        return (children[0]==NULL && children[1]==NULL && children[2]==NULL &&
            children[3]==NULL);
    }
```

```
    /// @brief Destruktor.
    ~QuadTreeNode(){
        for (size_t i=0; i<4; ++i)
            delete children[i];
        delete[] point;
    }
public:

    friend class QuadTree<T>;
    friend std::ostream& operator<< <T>(std::ostream &, const QuadTreeNode &);
};

/// @brief Milyen mélységig nőhet a fa? Ezzel a statikus változóval lehet
    beállítani.
template <class T>
unsigned QuadTreeNode<T>::MAX_LEVEL=10;

/// @brief Terület felbontása négy egybevágó téglalapra.
template <class T>
void QuadTreeNode<T>::split(){
    /// Bal felső.
    children[0]=new QuadTreeNode<T>(width/2, height/2, x, y+height/2, level+1,
        this);
    /// Jobb felső.
    children[1]=new QuadTreeNode<T>(width/2, height/2, x+width/2, y+height/2,
        level+1, this);
    /// Jobb alsó.
    children[2]=new QuadTreeNode<T>(width/2, height/2, x+width/2, y, level+1,
        this);
    /// Bal alsó.
    children[3]=new QuadTreeNode<T>(width/2, height/2, x, y, level+1, this);
}

/// @brief Új pont / adat beszúrása.
/// @param new_point - Új pont.
template <class T>
void QuadTreeNode<T>::insert(Point<T> new_point){
    /// Ha nincs meglévő adat:
    if (!hasData()){
        this->point = new Point<T>[1];
        point[0] = new_point;
        number_of_points=1;
    }
    /// Ha van meglévő adat és még nem érte el a maximális mélységet a fa:
    else if (hasData() && level != MAX_LEVEL){
        /// Felbontás.
        split();
        /// Beszúrjuk a meglévő adatot.
        size_t i=3;
        if (point->x <= x+width/2 && point->y > y+height/2) i=0;
        else if (point->x > x+width/2 && point->y >= y+height/2) i=1;
        else if (point->x > x+width/2 && point->y <= y+height/2) i=2;
        children[i]->insert(*point);
        delete[] point;
        point=NULL;
        number_of_points=0;
        /// Beszúrjuk az új adatot.
        i=3;
        if (new_point.x <= x+width/2 && new_point.y > y+height/2) i=0;
```

```
        else if (new_point.x > x+width/2 && new_point.y >= y+height/2) i=1;
        else if (new_point.x > x+width/2 && new_point.y <= y+height/2) i=2;
        children[i]->insert(new_point);
    }
    /// Ha van meglévő adat, de elérte a maximális mélységet a fa:
    else{
        /// Létre kell hozni egy pontokat tároló tömböt amiben egyel több
            elemet tárolhatunk el.
        Point<T> *temp = new Point<T>[number_of_points+1];
        size_t i;
        /// Átmásolja a meglévő elemeket az új tömbbe.
        for (i=0; i<number_of_points; ++i)
            temp[i]=point[i];
        /// Az új tömbbe berakja az új pontot.
        temp[number_of_points]=new_point;
        /// A csomópontban megnöveli a pontok számát tároló változót.
        ++number_of_points;
        /// A régi tömböt törli.
        delete[] point;
        point = temp;
    }
}

/// @brief QuadTree (generikus négy elágazású fa) osztály. A négy elágazású
    generikus fa QuadTreeNode-okból épül fel.
template <class T>
class QuadTree{
    /// Fa gyökerére mutató pointer.
    QuadTreeNode<T> *root;
public:
    /// @brief Konstruktor.
    QuadTree(double width=100, double height=100){
        root = new QuadTreeNode<T>(width, height);
    }

    /// @brief Destruktor.
    ~QuadTree(){delete root;}

    /// @return - Gyökérre mutató pointer.
    //QuadTreeNode<T>* getRootNode(){return root;}

    /// @brief Új elem beszúrása.
    /// @param data - Új pont / adat.
    void insert(Point<T> data);

    /// @brief Fa mélységének visszaadása.
    unsigned depth() const;

    /// @brief Fában lévő elemek megszámolása.
    /// @return Fa elemszáma.
    unsigned countNodes() const{
        unsigned nodes=0;
        for (iterator iter=begin(); iter!=end(); ++iter) ++nodes;
        return nodes;
    }

    /// @brief Adott pont megkeresése. Lehetőség van az adott pontban lévő
        adat megváltoztatására.
    Point<T>& find(Point<T> point);
```

```
    /// @brief Adott adatot tároló pont megkeresése. Lehetőség van az adott
        pontban lévő adat megváltoztatására.
    Point<T>& find(T data);

    /// @brief Fa iterátor.
    class iterator;
    /// @return - Legbaloldalibb levél.
    iterator begin() const{
        QuadTreeNode<T> *temp=root;
        while (!temp->isLeaf())
            temp=temp->children[0];
        return iterator(temp);
    }
    /// @return Az utolsó elem után mutat.
    iterator end() const{return iterator(NULL);}

    friend class QuadTreeNode<T>;
    friend std::ostream& operator<< <T>(std::ostream &, const QuadTree<T> &);
    /// Beolvasás. Az beolvasandó adatokat "(x;y): adat" alakban kapjuk.
    friend std::istream& operator>>(std::istream& is, QuadTree<T> & quadtree){
        double x, y;
        T data;
        while(is.good()){
            is.ignore(256, '(');
            is >> x;
            is.ignore(256, ';');
            is >> y;
            is.ignore(256, ')');
            is.ignore(256, ':');
            is >> data;
            is.ignore(256, '\n');
            quadtree.insert(Point<T>(data, x, y));
        }
        return is;
        }
};

/// @brief Fa mélységének visszaadása.
/// @return Fa mélysége.
template <class T>
unsigned QuadTree<T>::depth() const{
    if (root==NULL) return 0;
    unsigned max=0;
    for (iterator iter=begin(); iter!=end(); ++iter){
        if (max < (*iter).level)
            max=(*iter).level;
    }
    return max;
}

/// @brief Adott pont megkeresése. Lehetőség van az adott pontban lévő adat
    megváltoztatására.
/// @param point - Pont, melyet keresünk.
/// @return A keresett pont.
template <class T>
Point<T>& QuadTree<T>::find(Point<T> point){
    QuadTreeNode<T> *temp=root;
    while (!temp->isLeaf()){
```

```
        size_t i=3;
        if (point.x <= temp->x+temp->width/2 && point.y > temp->y+temp->height
            /2) i=0;
            else if (point.x > temp->x+temp->width/2 && point.y >= temp->y+
                temp->height/2) i=1;
                else if (point.x > temp->x+temp->width/2 && point.y <= temp->y
                    +temp->height/2) i=2;
                    temp=temp->children[i];
                    }
    for (size_t i=0; i<temp->number_of_points; ++i)
        if (point==temp->point[i]) return temp->point[i];
    throw "Point not found";
}

/// @brief Adott adatot tároló pont megkeresése. Lehetőség van az adott
    pontban lévő adat megváltoztatására.
/// @param data - Adat, melyet keresünk.
/// @return Az első olyan pont, amely ezt az adatot tartalmazza.
template <class T>
Point<T>& QuadTree<T>::find(T data){
    for (typename QuadTree<T>::iterator iter=QuadTree<T>::begin(); iter!=
        QuadTree<T>::end(); ++iter)
        for (size_t i=0; i<iter->number_of_points; ++i)
            if (iter->point[i].data==data)
                return find(Point<T>(data, iter->point[0].x, iter->point[0].y)
                    );
                throw "Point not found";
}

template <class T>
/// @brief Új elem beszúrása.
/// @param point - Új pont / adat.
void QuadTree<T>::insert(Point<T> point){
    /// Ha a pont kívül esik a területről, akkor kivételt dob.
    if (point.x > root->x+root->width || point.y > root->y+root->height)
        throw "This point can't be inserted.";
    /// Egyébként keressük meg azt a levelet ahova be kéne szúrni az új
        elemet.
    QuadTreeNode<T> *temp=root;
    while (!temp->isLeaf()){
        size_t i=3;
        if (point.x <= temp->x+temp->width/2 && point.y > temp->y+temp->height
            /2) i=0;
        else if (point.x > temp->x+temp->width/2 && point.y >= temp->y+temp->
            height/2) i=1;
        else if (point.x > temp->x+temp->width/2 && point.y <= temp->y+temp->
            height/2) i=2;
        temp=temp->children[i];
    }
    /// Szúrjuk be az elemet.
    temp->insert(point);
}

/// @brief Iterátor.
template <class T>
class QuadTree<T>::iterator{
protected:
    QuadTreeNode<T> *node;
public:
```

```
    /// @brief Konstruktor.
    iterator(QuadTreeNode<T> *node=NULL): node(node){}

    QuadTreeNode<T>& operator*(){return *node;}
    QuadTreeNode<T>* operator->(){return node;}

    /// @brief Prefix operator++
    /// @return Következő elem.
    iterator& operator++();

    /// @brief Postfix operator++
    iterator operator++(int){
        iterator temp(*this);
        ++(*this);
        return temp;
    }

    /// @brief Egyenlőség operátor.
    /// @return Igaz, ha megegyezik az csomópont.
    bool operator==(const iterator& iter){
        return node==iter.node;
    }

    /// @return Igaz, ha nem egyezik meg a csomópont.
    bool operator!=(const iterator& iter){
        return node!=iter.node;
    }
};

/// @brief Prefix inkrement.
/// @return Következő elem.
template <class T>
typename QuadTree<T>::iterator& QuadTree<T>::iterator::operator++(){
    size_t i=0;
    /// Ha a gyökér elemnél vagyunk (csak annak nincs szülője), akkor
       végigértünk az összes elemen.
    if (node->parent==NULL){
        node=node->parent;
        return *this;
    }
    /// A szülő hányadig gyerekén állunk?
    while (node!=node->parent->children[i])
        ++i;
    /// Ha a következő gyerek levél, akkor ez lesz a következő elem.
    if (i<3 && node->parent->children[i+1]->isLeaf())
        node=node->parent->children[i+1];
    /// Ha a következő gyerek nem levél, akkor a következő elem a
       legbaloldalibb levél a következő gyerek alatt.
    else if (i<3){
        QuadTreeNode<T> *temp=node->parent->children[i+1];
        while(!temp->isLeaf())
            temp=temp->children[0];
        node=temp;
    }
    /// Egyébként a szülő a következő elem.
    else
        node=node->parent;
    return *this;
}
```

```
/// @brief Point (pont) kiírása "(x;y): adat" alakban.
template <class T>
std::ostream& operator<<(std::ostream & os, const Point<T> & point){
    return os << '(' << point.x << ';' << point.y << ')' << ':' << ' ' <<
        point.data << std::endl;
}

/// @brief Fa kiírása iterátor használatával.
template <class T>
std::ostream& operator<<(std::ostream & os, const QuadTree<T> & quadtree){
    typename QuadTree<T>::iterator iter=quadtree.begin();
    while (iter!=quadtree.end()){
        os << *iter;
        ++iter;
    }
    return os;
}

/// @brief Csomópont kiírása.
template <class T>
std::ostream& operator<<(std::ostream & os, const QuadTreeNode<T> & node){
    /// Ha van adat, akkor ki kell írni.
    if (node.hasData()){
        for (size_t i=0; i<node.number_of_points; ++i){
            for (unsigned i=0; i<node.level; ++i) os << ' ';
            os << node.point[i];
        }
    }
    return os;
}
```