HÁZI FELADAT

Programozás alapjai 2.

Végleges

Borbola Martin

AC380P

2016. május 15.

**Tartalom**

[1. Feladat 1](#_Toc451109186)

[2. Pontosított feladatspecifikáció 1](#_Toc451109187)

[3. Terv 3](#_Toc451109188)

[3.1 Objektum terv 3](#_Toc451109189)

[3.2 Algoritmusok 3](#_Toc451109190)

[4. Tesztprogram algoritmusok 3](#_Toc451109191)

# Feladat

Programozás alapjai II. házi feladat

Borbola Martin (AC380P) részére:

Készítsen GENERIKUS bináris fát! A kulcsok közötti rendezettséget a szokásos relációs operátorokkal vizsgálja, amit szükség esetén specializál!

Valósítsa meg az összes értelmes műveletet operátor átdefiniálással (overload), de nem kell ragaszkodni az összes operátor átdefiniálásához! Amennyiben lehetséges használjon iterátort!

Demonstrálja a működést külön modulként fordított tesztprogrammal! A megoldáshoz NE használjon STL tárolót vagy algoritmust!

A tesztprogramot úgy specifikálja, hogy az parancssoros batch alkalmazásként (is) működjön, azaz a szabványos bemenetről olvasson, és a szabványos kimenetre, és/vagy a hibakimenetre írjon!

Amennyiben a feladat teszteléséhez fájlból, vagy fájlokból kell input adatot olvasnia, úgy a fájl neve \*.dat alakú legyen!

# Pontosított feladatspecifikáció

A feladat egy generikus bináris fa elkészítése. A fa típusát sablon paraméterként lehet megadni.

Az automatikusan létrejövő tagfüggvények mellett (másolás, értékadás, létrehozás, megszüntetés) keresést, beillesztést és a törlés műveletet valósítom meg, lehetőleg operátor átdefiniálással. Ezenkívül lehet fájlba menteni és onnan betölteni.

A teszteléséhez egy olyan programot készítek, ami különböző adattípusokkal létrehozott fákkal a standard inputról beolvasott parancsok alapján teszteket végez.

# Terv

A feladat egy objektum és a tesztprogram megtervezését igényli.

## Objektum terv

A generikus bináris fát két sablonnal fogom megvalósítani. Egy struktúraként a fa elemeit és egy osztályként az magát a fát. A sablon sablonparaméterként veszi át a fa elemeinek típusát.

Az iterator belső osztály, csak print(interator first, iterator last) függvénynél használom.

|  |
| --- |
| BSTNode |
| + data : T  + parent : BSTNode<T>\*  + left : BSTNode<T>\*  + right : BSTNode<T>\* |
| + operator<( BSTNode<T>)  + operator>( BSTNode<T>) |

|  |
| --- |
| BST |
| - root : BSTNode<T>\* |
| + BST()  + BST(ifstream)  + search(T) : bool  + insert(T) : void  + remove(T) : void  + sorban\_kiir() : void  + min() : iterator  + max() : iterator |

T: class

T: class

|  |
| --- |
| iterator |
| - curr: BSTNode<T>\* |
| + operator++() : iterator& // next  + operator++(int) : iterator  + operator--() : iterator& // prev  + operator--(int) : iterator  + operator\*() : T&  + operator->() : T\* |

## Algoritmusok

BST konstruktor: fájlból is lehet

BST::search: megkeresi az adott értékű elemet (rekurzívan)

BST::insert: beilleszt egy adott értékű elemet (rekurzívan)

BST::remove: kitörli az adott értékű elemet (rekurzívan)

BST::sorban\_kiiraz alapértelmezett kimenetre kiírja a fa elemit (iterátorral)

### Tesztprogram algoritmusai

A tesztprogram a standard inputról egy teszteset sorszámát olvassa be. Ez dönti el, hogy melyik teszteset fut. Minden teszteset mást tesztel.

# Megvalósítás

A tesztprogram beolvas fájlból egy bináris fát, majd ellenőrzi az algoritmusokat.

# Tesztelés