

# HÁZI FELADAT

## Pontosított feladatspecifikáció

Vizi Előd

S53O1S

2016. április 04.

---

**1.Feladat:** Az én feladatom készíteni egy objektumot 11 jegyű BCD számok tárolására. Az összes értelmes műveletet ezen az osztályon operátor átdefiniálással (overload) valósítam meg, viszont nem kell kötelezően minden operátort átdefiniálnom. Majd specifikálnom kell egy egyszerű tesztfeladatot, ami felhasználja ezt az adatszerkezetet.

**2.Pontosított feladatspecifikáció:** Mivel sokkal jobban szeretjük, legalább is én mindenképp jobban szeretem kezelni a decimális számokat, ezért mindenképp úgy valósítanám meg ezt az adatstruktúrát, hogy lenne két globális függvényem, amelyek meghívásával bármikor át tudom váltani a BCD számomat Decimálisra, valamint fordítva, Decimálisról vissza BCD –re. Továbbá így is valósítanám meg az operátorokat is. Két ilyen BCD összeadása esetén először mindkettőt átkonvertálnám decimálisra, majd úgy összeadom a számokat és végül pedig visszakonvertálnám BCD –re. Így könnyen megvalósíthatóvá válnak azok a műveletek is, amikor a művelet valamelyik oldalán mondjuk nem egy ilyen BCD kódolású szám áll, hanem egy decimális szeretnénk hozzáadni egy BCD –hez. Viszont a fent leírt konverzió nem valósítható meg integer változók segítségével. Ezért a biztonságos tárolás érdekében „long long int” –et szándékozom használni, hogy a 11 hosszú BCD –ket is tárolni tudjam decimálisként.

A feladat működésének szimulálására egy számológépet szeretnék bemutatni (grafikai felület nélkül) ami kimondottan ilyen adatszerkezetekkel dolgozik. Ez a számológép hatalmas segítséget nyújthat az első féléves Digitális Technika tantárgy hallgatói számára (és nemcsak számukra, hanem majdnem minden informatika érdekltségű ember számára, akik hasonló problémákba, gondolok én itt BCD számokkal végzett műveletekre, ütköznek munkájuk és tanulmányaik során).

**3.Terv:** Az én projektem egyetlen osztály felhasználásával megvalósítható, és nem is, hogy csak megvalósítható, hanem ez adja a lehető legegyszerűbb és legátláthatóbb megoldást is. Gondolkodtam azon is, hogy lenne egy osztályom ami a 4 bites szekvenciákat külön kezelne és a fő osztályom ezekből összefogólag kezelne maximum 11-et. De igazából ez a megvalósítás semmivel sem könnyítené meg az én munkámat sem, sem pedig az átláthatóságot, fejleszthetőséget. Így tehát úgy döntöttem, hogy egyetlen osztály segítségével oldom meg a feladatomat.

**3.1 Objektum Terv:** Ennek az osztálynak az UML diagramja a következő oldalon látható. Ezt a diagramot az „Enterprise Architect” nevű modellező program trial verziójával generáltam, ami letölthető az alábbi linken: <http://www.sparxsystems.com/products/ea/trial/request.html>.

---

Fontosnak tartom megjegyezni, hogy ez az osztály még bővíthet egy két operátorral, vagy esetleg egy-két újabb alprogrammal. Próbáltam átgondolni, hogy mikre lesz majd szükségem és az összes lehetséges felhasználandó operátort már most belerakni, de a fejlesztés folyamán adódó esetleges újabb problémák, egyszerűsítések megoldására, lehet szükséges lesz majd új függvényeket is bevezetni.

A legtöbb függvényt „friend” (barát) függvényként hoztam létre, hogy majd elérhetőek legyenek a függvénytörzsben az osztályom privát tagjai is.

class Class Model	
BCD	
-	darab: size_t
-	data: int*
+	BCD()
+	BCD(int)
+	BCD(BCD&)
+	~BCD()
+	BCD_to_llint(BCD&): long long int
+	llint_to_BCD(long long int): BCD
+	operator--(): BCD&
+	operator--(int): BCD&
+	operator=(BCD&): BCD&
+	operator=(long long int): BCD&
+	operator long long int() {query}
+	operator/=(long long int): BCD&
+	operator[](size_t): int {query}
+	operator[](size_t): int &
+	operator++(): BCD&
+	operator++(int): BCD&
+	operator+=(BCD&): BCD&
+	operator+=(long long int): BCD&
+	operator=(BCD&): BCD &
+	realsize(): size_t {query}
+	size(): size_t {query}
«friend»	
+	operator(BCD&, BCD&): BCD
+	operator(long long int, BCD&): BCD
+	operator(BCD&, long long int): BCD
+	operator!=(BCD&, BCD&): bool
+	operator!=(BCD&, long long int): bool
+	operator!=(long long int, BCD&): bool
+	operator<(BCD&, BCD&): bool
+	operator<(BCD&, long long int): bool
+	operator<(long long int, BCD&): bool
+	operator<=(BCD&, BCD&): bool
+	operator<=(BCD&, long long int): bool
+	operator<=(long long int, BCD&): bool
+	operator==(BCD&, BCD&): bool
+	operator==(BCD&, long long int): bool
+	operator==(long long int, BCD&): bool
+	operator>(BCD&, BCD&): bool
+	operator>(BCD&, long long int): bool
+	operator>(long long int, BCD&): bool
+	operator>=(BCD&, BCD&): bool
+	operator>=(BCD&, long long int): bool
+	operator>=(long long int, BCD&): bool
+	operator%(BCD&, long long int): BCD
+	operator%(long long int, BCD&): BCD
+	operator&(BCD&, BCD&): BCD
+	operator*(BCD&, BCD&): BCD
+	operator/(BCD&, BCD&): BCD
+	operator/(BCD&, long long int): BCD
+	operator^(BCD&, BCD&): BCD
+	operator (BCD&, BCD&): BCD
+	operator+(BCD&, BCD&): BCD
+	operator+(long long int, BCD&): BCD
+	operator+(BCD&, long long int): BCD
+	operator<<(std::ostream&, BCD&): std::ostream&
+	operator>>(std::istream&, BCD&): std::istream&

**3.2 Algoritmusok:** A tagfüggvények definiálása és az osztályom létrehozása folyamán pár szükséges és alapvető függvényt már a létrehozás folyamán megírtam, ilyenek például a konstruktor, destruktork, operator=, valamint az indexelő (operator[], inline függvényekként) operátorok. Ezek az algoritmusok alább láthatók. Azonban ezek is még változhatnak a fejlesztés folyamán.

```
#include<iostream>
#include<cmath>
#include "BCD.h"
#define HOSSZ darab*4 //darab 4 szerese, mivel egy tagot 4 bitbe tarol

BCD::BCD() {
    data = NULL;
    darab=0;
}

BCD::BCD(int darab):darab(darab) {
    data=NULL;
    if(darab!=0) {
        data=new int[HOSSZ];
        for(int i=0;i<HOSSZ;i++) {
            data[i]=0;
        }
    }
}

BCD::BCD(const BCD & rhs){
    darab=rhs.darab;
    if(rhs.data==NULL)
        data=NULL;
    else{
        data = new int[HOSSZ];
        for(int i=0;i<HOSSZ;i++){
            data[i]=rhs.data[i];
        }
    }
}

BCD& BCD::operator= (const BCD & rhs){
    if(this!=&rhs){
        delete[] data;
        darab=rhs.darab;
        data=new int[HOSSZ];
        for(int i=0;i<HOSSZ;i++){
            data[i]=rhs.data[i];
        }
    }
    return *this;
}
```

```
int operator[] (size_t idx) const{
    return data[idx];
}

int & operator[] (size_t idx){
    return data[idx];
}

size_t size() const{
    return darab;
}

size_t realsize() const{
    return HOSSZ;
}
```

My Project - BCD class

Készítette Doxygen 1.8.11



# Tartalomjegyzék

<b>1. Osztálymutató</b>	<b>1</b>
1.1. Osztálylista	1
<b>2. Osztályok dokumentációja</b>	<b>3</b>
2.1. BCD osztályreferencia	3
2.1.1. Részletes leírás	4
2.1.2. Konstruktorkok és destruktorkok dokumentációja	4
2.1.2.1. BCD(int darab=0)	4
2.1.2.2. BCD(int *data, int darab)	5
2.1.2.3. BCD(const BCD &rhs)	5
2.1.2.4. ~BCD()	5
2.1.3. Tagfüggvények dokumentációja	5
2.1.3.1. bcd_to_dec()	5
2.1.3.2. operator!=(BCD &) const	5
2.1.3.3. operator!=(long long int) const	5
2.1.3.4. operator>(long long int) const	6
2.1.3.5. operator>=(BCD &) const	6
2.1.3.6. operator>=(long long int) const	6
2.1.3.7. operator[](size_t idx) const	7
2.1.3.8. operator[](size_t idx)	7
2.1.3.9. realsize() const	7
2.1.3.10. size() const	7
2.1.4. Barát és kapcsolódó függvények dokumentációja	8
2.1.4.1. operator&	8
2.1.4.2. operator<<	8
2.1.4.3. operator>>	8
2.1.4.4. operator^	8
2.1.4.5. operator"	8



# 1. fejezet

## Osztálymutató

### 1.1. Osztálylista

Az összes osztály, struktúra, unió és interfész listája rövid leírásokkal:

BCD	BCD osztály . . . . .	3
-----	-----------------------	---





## 2. fejezet

# Osztályok dokumentációja

### 2.1. BCD osztályreferencia

BCD osztály.

```
#include <BCD.h>
```

#### Publikus tagfüggvények

- `BCD` (int `darab`=0)
- `BCD` (int `*data`, int `darab`)
- `BCD` (const `BCD` &`rhs`)
- `BCD & operator=` (const `BCD` &`rhs`)
- int `operator[]` (size\_t `idx`) const
- int & `operator[]` (size\_t `idx`)
- size\_t `size` () const
- size\_t `realize` () const
- long long int `bcd_to_dec` ()
- `BCD & dec_to_bcd` (long long int `szam`)
- `BCD operator+` (`BCD` &) const
- `BCD operator+` (long long int) const
- `BCD operator-` (`BCD` &) const
- `BCD operator-` (long long int) const
- `BCD & operator+=` (`BCD` &)
- `BCD & operator+=` (long long int)
- `BCD & operator-=` (`BCD` &)
- `BCD & operator-=` (long long int)
- `BCD operator*` (`BCD` &)
- `BCD operator/` (`BCD` &)
- `BCD operator/` (int)
- `BCD operator/` (long long int)
- `BCD operator/` (double)
- `BCD operator%` (`BCD` &)
- `BCD operator%` (int)
- `BCD operator%` (long long int)
- `BCD & operator/=` (`BCD` &)
- `BCD & operator/=` (long long int)

- bool `operator==` (`BCD &`) const
- bool `operator!=` (`BCD &`) const
- bool `operator<` (`BCD &`) const
- bool `operator>` (`BCD &`) const
- bool `operator<=` (`BCD &`) const
- bool `operator>=` (`BCD &`) const
- bool `operator==` (long long int) const
- bool `operator!=` (long long int) const
- bool `operator<` (long long int) const
- bool `operator>` (long long int) const
- bool `operator<=` (long long int) const
- bool `operator>=` (long long int) const
- `BCD & operator++` ()
- `BCD & operator--` ()
- `BCD & operator++` (int)
- `BCD & operator--` (int)
- `~BCD` ()

### Védett attribútumok

- size\_t `darab`  
*BCD-ként tárolt szám számjegyeinek a száma.*
- int \* `data`  
*Az adott számot binárisan (BCD-ként) tároló tömb.*

### Barátok

- `BCD operator&` (const `BCD &`, const `BCD &`)
- `BCD operator|` (const `BCD &`, const `BCD &`)
- `BCD operator^` (const `BCD &`, const `BCD &`)
- std::ostream & `operator<<` (std::ostream &os, const `BCD &rhs`)
- std::istream & `operator>>` (std::istream &is, `BCD &rhs`)

## 2.1.1. Részletes leírás

`BCD` osztály.

## 2.1.2. Konstruktorkok és destruktorkok dokumentációja

### 2.1.2.1. `BCD::BCD ( int darab = 0 )`

Konstruktork - méret alapján

Paraméterek

<code>darab</code>	- számjegyek száma (tömb méretének negyede)
--------------------	---

2.1.2.2. `BCD::BCD ( int * data, int darab )`

Konstruktor - 0, 1 számokat tartalmazó tömbből

Paraméterek

<i>darab</i>	- számjegyek száma (tömb méretének negyede)
<i>*data</i>	- integer tömb

2.1.2.3. `BCD::BCD ( const BCD & rhs )`

Copy/Másoló konstruktor

Paraméterek

<i>rhs</i>	- jobb oldali operandus(BCD)
------------	------------------------------

2.1.2.4. `BCD::~~BCD ( ) [inline]`

Destruktor Mivel egy dinamikus tömböt tárolunk, ezért gondoskodni kell a felszabadításáról is.

## 2.1.3. Tagfüggvények dokumentációja

2.1.3.1. `long long int BCD::bcd_to_dec ( )`

**BCD** típus atkasztolasa egész számá (hossza miatt, 11 jegyû is lehet akár) long long int Az egész szám (long long int) atkasztolasa **BCD** be

2.1.3.2. `bool BCD::operator!= ( BCD & rhs ) const`

Egyenlőtlenség vizsgálat

Paraméterek

<i>rhs</i>	- jobb oldali operandus ( <b>BCD</b> )
------------	--

Visszatérési érték

true - ha a két **BCD** nem egyenlő, egyébként false

2.1.3.3. `bool BCD::operator!= ( long long int szam ) const`

Egyenlőtlenség vizsgálat

## Paraméterek

<i>szam</i>	- jobb oldali operandus (long long int)
-------------	---

## Visszatérési érték

true - ha a két adat nem egyenlő, egyébként false

## Paraméterek

<i>rhs</i>	- jobb oldali operandus (BCD)
------------	-------------------------------

## Visszatérési érték

true - ha a jobb oldali BCD kisebb mint a bal oldali, egyébként false

2.1.3.4. bool BCD::operator> ( long long int *szam* ) const

vizsgálat

## Paraméterek

<i>szam</i>	- jobb oldali operandus (long long int)
-------------	---

## Visszatérési érték

true - ha a jobb oldali adat kisebb mint a bal oldali, egyébként false

2.1.3.5. bool BCD::operator>= ( BCD & *rhs* ) const

>= vizsgálat

## Paraméterek

<i>rhs</i>	- jobb oldali operandus (BCD)
------------	-------------------------------

## Visszatérési érték

true - ha a jobb oldali BCD kisebb vagy egyenlő mint a bal oldali, egyébként false

2.1.3.6. bool BCD::operator>= ( long long int *szam* ) const

>= vizsgálat

## Paraméterek

<i>szam</i>	- jobb oldali operandus (long long int)
-------------	---

## Visszatérési érték

true - ha a jobb oldali adat kisebb vagy egyenlő mint a bal oldali, egyébként false

2.1.3.7. `int BCD::operator[] ( size_t idx ) const` `[inline]`

Index operator - konstans

## Paraméterek

<i>idx</i>	- index
------------	---------

## Visszatérési érték

indexedik elem

2.1.3.8. `int& BCD::operator[] ( size_t idx )` `[inline]`

Index operator - nem konstans, módosítható esetben

## Paraméterek

<i>idx</i>	- index
------------	---------

## Visszatérési érték

indexedik elem

2.1.3.9. `size_t BCD::realsize ( ) const` `[inline]`

Tárolt data-tömb konkrét méretének lekérdezése

## Visszatérési érték

- data tömb mérete

2.1.3.10. `size_t BCD::size ( ) const` `[inline]`

Méret lekérdezése

## Visszatérési érték

- számjegyek száma

## 2.1.4. Barát és kapcsolódó függvények dokumentációja

### 2.1.4.1. BCD operator& ( const BCD & , const BCD & ) [friend]

Bitenként ÉS-eli a két BCD-t

### 2.1.4.2. std::ostream& operator<< ( std::ostream & os, const BCD & rhs ) [friend]

Kiírás egy ostream-re

Paraméterek

<i>os</i>	- bal oldali operandus (ostream)
<i>rhs</i>	- jobb oldali operandus (BCD)

Visszatérési érték

ostream&, hogy fűzhető legyen

### 2.1.4.3. std::istream& operator>> ( std::istream & is, BCD & rhs ) [friend]

Beolvasás egy istream-ről

Paraméterek

<i>is</i>	- bal oldali operandus (istream)
<i>rhs</i>	- jobb oldali operandus (BCD)

Visszatérési érték

istream&, hogy fűzhető legyen

### 2.1.4.4. BCD operator^ ( const BCD & , const BCD & ) [friend]

Bitenkénti kizáró VAGY két BCD között (XOR)

### 2.1.4.5. BCD operator| ( const BCD & , const BCD & ) [friend]

Bitenként VAGY-olja a két BCD-t (megengedő vagy)

Ez a dokumentáció az osztályról a következő fájlok alapján készült:

- BCD.h
- BCD.cpp

# Tárgymutató

~BCD

BCD, 5

BCD, 3

~BCD, 5

BCD, 4, 5

bcd\_to\_dec, 5

operator!=, 5

operator<<, 8

operator>, 6

operator>>, 8

operator>=, 6

operator^, 8

operator%, 6

operator&, 8

operator[], 7

operator|, 8

realsize, 7

size, 7

bcd\_to\_dec

BCD, 5

operator!=

BCD, 5

operator<<

BCD, 8

operator>

BCD, 6

operator>>

BCD, 8

operator>=

BCD, 6

operator^

BCD, 8

operator%

BCD, 6

operator&

BCD, 8

operator[]

BCD, 7

operator|

BCD, 8

realsize

BCD, 7

size

BCD, 7