Katona József–Király Zoltán

OBJEKTUMORIENTÁLT SZOFTVERFEJLESZTÉS C++ NYELVEN

KONSTRUKTOR TÍPUSOK, DESTRUKTOR, OSZTÁLYSABLONOK, FÜGGVÉNYSABLONOK ÉS FÜGGVÉNYTÚLTERHELÉSEK

A konstruktor típusok

A destruktor avagy az objektumpéldány megszüntetése

Paraméterezett típusok

Dunakavics
D·U·F PRESS

INFORMATIKAI KÖNYVEK

2.

Katona József–Király Zoltán

OBJEKTUMORIENTÁLT SZOFTVERFEJLESZTÉS C++ NYELVEN

(KONSTRUKTOR TÍPUSOK, DESTRUKTOR, OSZTÁLYSABLONOK, FÜGGVÉNYSABLONOK ÉS FÜGGVÉNYTÚLTERHELÉSEK)

Dunaújváros, 2015

© Katona József–Király Zoltán, 2015

Lektorálta: Dr. Kusper Gábor, tanszékvezető főiskolai docens

D▪U▪F PRESS
DUNAÚJVÁROSI FŐISKOLA
www.duf.hu

Kiadóvezető: Németh István
Felelős szerkesztő: Nemeskéry Artúr
Layout és tördelés: Duma Attila

Kiadja: DUF Press, a Dunaújvárosi Főiskola kiadója
Felelős kiadó: Dr. András István

Készült a HTSART nyomdában.
Felelős vezető: Halász Iván

ISBN 978-963-9915-53-4
ISSN 2415-9115

Katona József–Király Zoltán

OBJEKTUMORIENTÁLT SZOFTVERFEJLESZTÉS C++ NYELVEN

KONSTRUKTOR TÍPUSOK, DESTRUKTOR, OSZTÁLYSABLONOK, FÜGGVÉNYSABLONOK ÉS FÜGGVÉNYTÚLTERHELÉSEK

A konstruktor típusok

A destruktor avagy az objektumpéldány megszüntetése

Paraméterezett típusok

Dunakavics
D·U·F PRESS
Dunaújváros, 2015

INFORMATIKAI KÖNYVEK

2.

Tartalom

1. Fejezet

A konstruktor típusok 9

Alapértelmezett konstruktor (default constructor) 10

Másoló konstruktor (copy constructor) 10

Mintapélda 11

A destruktor avagy az objektumpéldány megszüntetése 13

Mintapélda 14

Paraméterezett típusok 15

Függvénysablonok, függvény felüldefiniálások 15

Mintapélda 16

Osztálysablonok 19

Mintapélda 19

Gyakorló példák 20

2. Forráskódok

Művelet osztály deklarációja (Muvelet.h) 25

Művelet osztály implementálása (Muvelet.cpp) 26

Művelet osztály példányosítása (main.cpp) 27

Rectangle osztály deklarációja (Rectangle.h) 28

Rectangle osztály példányosítása (main.cpp) 30

Swap osztály deklarációja (Swap.h) 30

Swap osztály implementálása (Swap.cpp) 31
Swap osztály példányosítása (main.cpp) 32
OSwap osztály deklarációja (OSwap.h) 33
OSwap osztály implementálása (OSwap.cpp) 34
OSwap osztály példányosítása (main.cpp) 35
TSwap osztály deklarációja (TSwap.h) 36
TSwap osztály implementálása (TSwap.cpp) 36
TSwap osztály példányosítása (main.cpp) 37
TGetMax osztály deklarációja (TGetMax.h) 38
TGetMax osztály implementálása (TGetMax.cpp) 38
TGetMax osztály példányosítása (main.cpp) 39

3. Felhasznált irodalom 40

Előszó

Úgy gondoljuk, hogy ez a könyvsorozat hozzájárulhat ahhoz is, hogy különböző szakemberek vagy akár középiskolai diákok is kamatoztathassák tudásukat a témában.

A műsorozat elsődleges célja az objektumorientált világ megismertetése. Az ismeretek elsajátításhoz igyekeztünk egy olyan magas szintű programozási nyelvet választani, amely az előismeretekre támaszkodva hatékonyan képes szemléltetni az OOP világában használt szabályokat, fogalmakat, elveket és modelleket. A választott C++ nyelv mindamellett, hogy a fentebb leírt előírásoknak megfelel, a munkaerőpiacon is az egyik legsikeresebbnek számító magasszintű nyelv.

A szoftverfejlesztési és annak tanítási tapasztalataink alapján arra az elhatározásra jutottunk, hogy a választott C++ nyelvet nem a legelejétől kívánjuk ismertetni. Nem szeretnénk például elismételni azokat a vezérlési szerkezeteket, amelyeket már egy középszinten lévő programozó számtalanszor hallott és jó eséllyel használt. Természetesen ez nem azt jelenti, hogy ezeket az eszközöket el is felejthetjük, ugyanis, ahhoz, hogy jó programozó váljon belőlünk nem elég ismernünk ezeket az utasításkészleteket, hanem fel is kell ismernünk, hogy hol és mikor a legcélszerűbb azokat használni.

A könyv összeállításakor az ipari és oktatói tapasztalatainkra és számos, neves szakirodalomra támaszkodtunk. Reméljük, hogy ez a mű hozzásegíti az olvasót a tananyag megértéséhez és könnyed elsajátításához. Ehhez kellő kitartást és lelkesedést kívánunk.

Köszönettel tartozunk a kézirat lektorának, Dr. Kusper Gábornak gondos munkájáért, szakmai tanácsaiért és módszertani javaslataiért, melyek jelentősen hozzájárultak a könyvsorozat magasabb szakmai szintre emeléséhez.

A Szerzők

OOP: angolul object-oriented programming egy programozási módszertan

KATONA JÓZSEF *–KIRÁLY ZOLTÁN **

Objektumorientált programozás C++ nyelven

(Konstruktor típusok, Destruktor, Osztálysablonok, Függvénysablonok és Függvénytúlterhelések)

* *Dunaújvárosi Főiskola, Informatika Intézet*
E-mail: katonaj@mail.duf.hu

** *Dunaújvárosi Főiskola, Informatika Intézet*
E-mail: kiru@mail.duf.hu

1. Fejezet

A konstruktor típusok

A konvencionális programozási nyelvek széles körű problémája az úgynevezett inicializálás vagy más néven kezdetiérték-adás. A programozó által létrehozott adattípusoknak ugyanolyan integráns részei egy programnak, mintha beépített típusok lennének, ez a C++ filozófiájának egyik lényeges eleme. Ebből kiindulva a C++ tervezői és megalkotói egy speciális függvényt hoztak létre, amely egyértelműen meghatározzák, hogy milyen módon fogalmazhatóak meg egy adott osztály objektumai. Ezt a speciális függvényt konstruktornak nevezzük.
Az eddigi programok megoldása során, ha szükség volt konstruktorra, akkor annak a paraméteres alakját használtuk, ahol a konstruktor a paraméterben kapott értékeket adta értékül az osztály adattagjainak.

Az előző könyvben a konstruktort egy olyan speciális függvénynek tekintettük, melynek a törzsében lévő kód akkor fut le, amikor az adott osztályból létrehozunk egy példányt. A konstruktor általános feladata az osztály adattagjainak inicializálása. A legfontosabb tulajdonságai a következőek:
– A neve meg kell, hogy egyezzen az Osztály nevével.
– Nincs visszatérési értéke, még void sem.
– Mint minden más metódus a konstruktor is túlterhelhető, így eltérő paraméterezésű konstruktor is létrejöhet.
– Ha nincs általunk létrehozott konstruktor, akkor a fordító elkészíti az alapértelmezett konstruktort.
– Nem öröklődhet.

Az adattagok inicializálása mellett gyakorta további feladatokat is ellát. Az olyan osztályoknál, amelyek memóriaigénye nagyobb, elengedhetetlen a szükséges memóriaterület lefoglalása.

Alapértelmezett konstruktor (default constructor)

A paraméteres konstruktor mellett létezik paraméter nélküli is, amelyet alapértelmezett konstruktornak (default) nevezünk. Az ilyen típusú konstruktor törzsében általában az osztály adattagjait látjuk el konstans kezdőértékkel. Az alábbi példa egy alapértelmezett konstruktor felépítését illusztrálja, ahol egy osztály (Komplex) két változóját (re, im) konstans nulla értékkel látjuk el:

```
Komplex :: Komplex()
{
        this->re=0; this->im=0;
}
```

Másoló konstruktor (copy constructor)

A konstruktorok világában létezik egy harmadik típus, amelyet másoló konstruktornak nevezünk (copy constructor). Ezt a függvénytípust használva egy objektumpéldánynak, kezdőértéket adhatunk egy már létező és inicializált másik objektumpéldánnyal. A másolókonstruktor is képes arra, amire az összes többi: létrehozáskor inicializálhatjuk vele az objektumainkat. A másoló konstruktor paraméterének nélkülözhetetlen velejárója, hogy az r*eferencia* legyen, különben végtelen ciklus keletkezik, mert a paraméterátadáshoz is a másoló konstruktort használjuk. Az alábbiakban láthatunk egy példát a másoló konstruktor implementálására, ahol a Komplex osztályhoz tartozó másolókonstruktor paramétere egy ugyanolyan típusú refe-rencia:

```
Komplex :: Komplex(const Komplex &object)
{
        this->re=object.re; this->im=object.im;
}
```

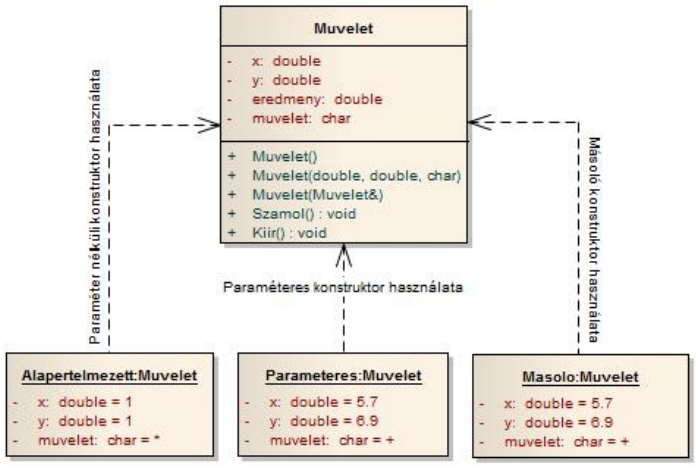
Az osztály nem csak egy konstruktort tartalmazhat, hanem többet is, és mindig az argumentumlista alapján dől el, hogy melyik változat kerül felhasználásra.

Mintapélda

Írjunk egy Művelet osztályt, ahol alapértelmezett, paraméteres és másoló konstruktor is szerepel. Az osztály feladata a négy alapvető művelet közül az egyik elvégzése és eredményének kiíratása. Az alapértelmezett konstruktornál válasszuk az összeadást, a paraméteres konstruktornál a felhasználó döntse el, hogy milyen műveletet szeretne végezni. A másolókonstruktor feladata pedig, hogy a paraméteres példánynál felhasznált értékeket másolja le és végezze el ugyanazt a műveletet. (CA_03)

Osztálydiagram (Class diagram)

Objektumdiagram (Object diagram)



Az osztály deklarációját a *(Muvelet.h)* állományban adjuk meg. A *Muvelet* osztály magán adattagjai:
– double *x,y* az operandusok;
– double *eredmeny* tárolja a művelet eredményét;
– char *muv* a műveleti jelet tartalmazza;

A *Muvelet* osztály nyilvános tagfüggvényei:
– A művelet végrehajtását, azaz a számítást a *Szamol()* tagfüggvény végzi:

```
void Szamol();
```

– Ennek a void típusú függvénynek nincs paraméterlistája, hiszen az objektum adattagjait használja a műveletvégzéshez. A *Szamol()* tagfüggvény feladata, hogy a *muv* operátor alapján kiválasztott műveletet végrehajtsa, és az eredményt az *eredmeny* adattagban tárolja.
– Az eredmény közlése a *Kiir()* tagfüggvény feladata:

```
void Kiir();
```

– A *Kiir()* tagfüggvénynek az adattagokban tárolt adatokat írja ki szöveges formában.

Három eltérő típusú konstruktort készítettünk:
– A paraméter nélkülit, ahol az adattagokat konstans kezdőértékkel látjuk el:

```
Muvelet() {x=1; y=1; muv='+';}
```

– A paraméterest, ahol a paraméterben kapott értékekkel látjuk el az osztály adattagjait.
```
Muvelet(double x1, double y1, char muv1);
```
– Másoló konstruktort, ahol egy másik objektumpéldány értékeivel látjuk el az objektumpéldányt.
```
Muvelet(const Muvelet &v);
```
A *(Muvelet.cpp)* állomány tartalmazza az osztály tagfüggvényeinek kidolgozását. A tagfüggvény neve előtt a :: hatókör-operátort használjuk arra, hogy jelezzük melyik osztály tagfüggvényét akarjuk kifejteni.

A *(main.cpp)* állomány tartalmazza a main() függvényt. A függvény kifejtése előtt szükséges a *(Muvelet.h)* fejlécállomány beépítése, hiszen benne találhatóak a *Muvelet* osztály deklarációi. A főfüggvényben történő példányosítás után hívjuk meg a szükséges függvényeket. Az **strchr()** függvény az *m* karakter +-*/ beli első előfordulási helyének mutatójával, illetve, ha *m* nem található meg +-*/-ban, akkor NULL értékű „mutatóval” tér vissza.

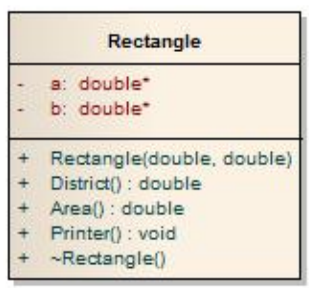
A destruktor avagy az objektumpéldány megszüntetése

Az eddigi példákban a memóriát, mint statikus eszközt kezeltük. Ez azt jelentette, hogy az objektumpéldányt létrehoztuk, de a példány megszüntetését magára az operációs rendszerre bíztuk. Tehát a destruktor, az objektum megszüntetésért, a memória felszabadításáért felelős, amely eddig automatikusan került meghívásra. Azonban vannak olyan esetek, amikor a memóriát célszerűbb dinamikusan kezelni, tehát maga a programozó szabja meg, hogy egy objektumpéldány meddig létezzen. Néhány fontosabb tulajdonság, amely igaz a destruktorra:
– A destruktor nevét a hullám karakterrel (~) kell kezdeni.
– A konstruktorhoz hasonlóan a neve megegyezik az osztály nevével.
– A konstruktorhoz hasonlóan itt sincs vissza térési érték, még *void* sem.
– Ha az osztály rendelkezik destruktorral, akkor azt hívja meg a fordító, ha azonban ilyen nem létezik, akkor az adott osztályhoz a fordító a saját alapértelmezése szerinti változatát használja.
– A destruktor aktivizálása a *delete* operátorral történik.
– Egy destruktor lehet virtuális is, azonban egy konstruktor nem.

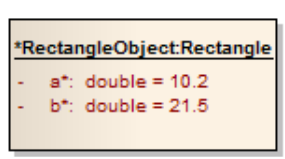
Mintapélda

Készítsünk osztályt, melynek feladata egy téglalap kerületének és területének a kiszámolása. A memóriát dinamikusan kezeljük. (CA_04)

Osztálydiagram (Class diagram)



Objektumdiagram (Object diagram)



A *(Rectangle.h)* állomány tartalmazza az osztály deklarációit, azonban itt a függvények is kifejtése kerülnek, ezért külön implementációs fájlra nincs szükség. Az **include** fájlokra a már fentebb megírt programoknál használt műveletek miatt van szükség. Ahhoz, hogy a **cout** és a **cin** függvényeket elérjük, szükség van az úgynevezett **std** névtérre. Ennek behívása történik a **using** kulcsszóval. Az osztályunk neve *Rectangle*.
A példánkban a téglalap magasságát és szélességét adattag-mutatókban tároljuk. Az adattag-mutatók egyszeres indirektségű mutatóként viselkednek, azaz egy memóriacímre hivatkoznak.

Az osztály konstruktora a **new** operátor segítségével lefoglalja a memóriaterületet a két adattag-mutató számára, ahol átadásra kerül a téglalap magassága és szélessége. Az osztály két egyszerű feladatát a *District()* és az *Area()* függvény látja el, továbbá a konzolos információ megjelenítésért a *Printer()* függvény felel. Az utolsó függvényünk a destruktor, amely majd a memóriaterület felszabadításáért lesz felelős, illetve további feladata az adattag-mutatók által foglalat memóriaterület felszabadítása, amely a **delete** operátorral történik. Abban az esetben, ha a függvény feladatát nem az osztály deklarációján kívül fejtjük ki **inline** módszernek nevezzük. Az **inline** módszert kisterjedelmű függvényeknél érdemes használni, mert ebben az esetben olyan, mintha makró lenne, tehát a függvényhívásokkal járó adminisztrációra nincs szükség. Rövid, egy-két soros függvények esetében ez sokkal hatékonyabb, sokkal olvashatóbb megoldást jelent. A fő függvényünkben *(main.cpp)* a *Rectangle* egy példányát (objektumát) dinamikusan hozzuk létre. Mi saját magunk memóriát foglalunk le az objektum (példány) részére, és már lehet is vele műveletet végezni. A **new** operátorral hívjuk a konstruktort és adjuk át neki az értékeket. Az osztály **public** láthatóságú tagjaira „->” operátorral hivatkozunk. A példány megsemmisülése általunk vezérelten fog történni, nem lesz automatikus. Az objektum mindaddig foglalt marad, amíg azt fel nem szabadítjuk a **delete** operátor segítségével. Egy osztályból tetszőleges példányt (ameddig a memória bírja) hozhatunk létre.

Paraméterezett típusok

A sablonok (template) használata lehetővé teszi, hogy egymással logikai kapcsolatban álló függvények és osztályok családját hozzuk létre. A C++ sablonok tulajdonképpen a generikus típusok C++ nyelvbeli megfelelői. Az osztályok és függvények akkor válnak teljessé, ha azok hívásra kerülnek. A nagy keretrendszer-fejlesztők szeretik a sablonokat, ugyanis, ha például egy függvényt fel szeretnénk használni több típussal (**integer, double, char, string, bool** stb.), akkor minden egyes típusnál új metódust kell írnunk, azonban a sablonok használatával elég azt csak egyszer megírnunk és a híváskor átadni neki azt a típust, amivel dolgozni szeretnénk.

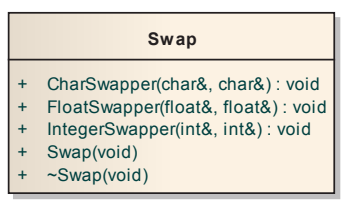
Függvénysablonok, függvény felüldefiniálások

Ha olyan függvényt szeretnénk használni, ami különböző típusú paraméterekkel dolgozik, akkor azt úgy tehetjük meg, hogy külön minden típushasználatakor írunk egy új függvényt. Egy másik sokkal szebb megoldás a függvény-felüldefiniálás, ahol is az átdefiniálás során a függvény törzse szinte nem változik. Továbbá a C++ lehetőséget ad úgynevezett *sablonok* használatára, ahol is elég a függvényt egyszer megírni és a továbbiakban az bármilyen típussal el fog boldogulni.

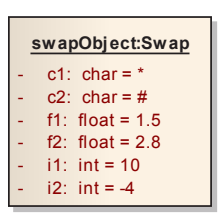
Mintapélda

Az eddig megismert fogalmak segítségével oldjunk meg egy olyan feladatot, ahol adatcsere-függvények fognak szerepelni, amelyek felcserélnek két egész, valós és karaktertípusú változó tartalmát. (CA_05) A továbbiakban ezt a problémát oldjuk meg függvény-túlterheléssel. (CA_06) Végezetül nézzük meg, hogyan is nézne ki a feladat függvénysablon segítségével. (CA_07)

Osztálydiagram (Class diagram)

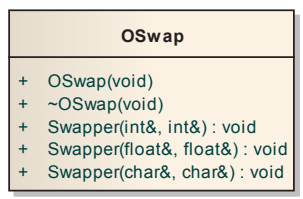


Objektumdiagram (Object diagram)

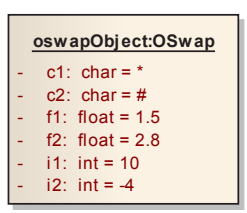


Az osztály deklarációját a *(Swap.h)* állományban adjuk meg. A különböző típusú adatok cseréjéhez külön függvényeket írunk, amelyek a paraméterben kapott értékek csere-végrehajtásáért lesznek felelősek. A *(Swap.cpp)* állomány tartalmazza az osztály tagfüggvényeinek a kidolgozását. A tagfüggvény neve előtt a :: hatókör-operátort használjuk arra, hogy jelezzük melyik osztály tagfüggvényét óhajtjuk kifejteni. A *(main.cpp)* állomány tartalmazza a **main()** függvényt. A függvény előtt szükséges a *(Swap.h)* fejlécállomány beépítése, hiszen benne találhatóak a *Swap* osztály deklarációi. A **main**-ben történő példányosítás után hívjuk meg a szükséges függvényeket. Nézzünk meg egy másik megoldási módszert, ahol a függvénytúlterhelést fogjuk alkalmazni.

Osztálydiagram (Class diagram)



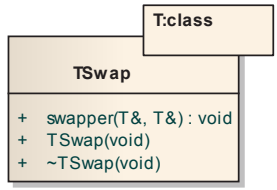
Objektumdiagram (Object diagram)



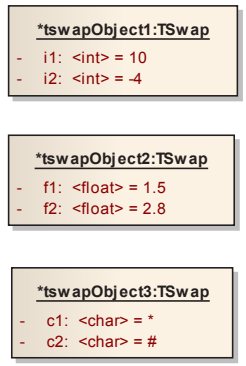
Az osztály deklarációját az *(OSwap.h)* állományban adjuk meg. A különböző típusú adatok cseréjéhez külön függvényeket írtunk, amelyek a paraméterben kapott értékek cserevégrehajtásáért felelősek. A különbség viszont az, hogy itt már azonos nevű függvényeket használunk. Az *(OSwap.cpp)* állomány tartalmazza az osztály tagfüggvényei kidolgozását. A tagfüggvény neve előtt a :: hatókör-operátort használjuk arra, hogy jelezzük melyik osztály tagfüggvényét szeretnénk kifejteni. A *(main.cpp)* állomány tartalmazza a **main()** függvényt. A függvény előtt szükséges az *(OSwap.h)* fejlécállomány beépítése, hiszen benne találhatóak az OSwap osztály deklarációi. A **main**ben történő példányosítás után hívjuk meg a szükséges függvényeket.

A következő példában az adatcsere függvény paramétereinek sablonnal történő megoldása szerepel.

Osztálydiagram (Class diagram)



Objektumdiagram (Object diagram)



Az osztály deklarációját a *(TSwap.h)* állományban adjuk meg. A szükséges paramétereket az osztályok és a tagfüggvények elé kell beszúrni, valamint a paraméterek típusát is jelölni kell. A csere végrehajtásához itt már csak egyetlen függvényt kell megírnunk és az típustól függetlenül tud feladatot végrehajtani. A *(TSwap.cpp)* állomány tartalmazza az osztály tagfüggvényeinek a kidolgozását. A tagfüggvény neve előtt a :: hatókör-operátort használjuk arra, hogy jelezzük melyik osztály tagfüggvényét óhajtjuk kifejteni. A *(main.cpp)* állomány tartalmazza a **main()** függvényt. Itt az osztályon kívül fejtjük ki a függvényeket és a sablonok használata miatt nem a **.h** kiterjesztésű fájlt kell behívni, hanem a **.cpp** kiterjesztésűt. A **main**ben történő példányosítás után hívjuk meg a szükséges függvényeket. A példány megszüntetéséhez használjuk a delete operátort.

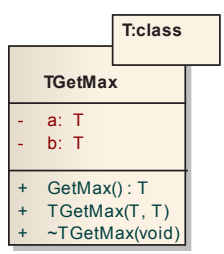
Osztálysablonok

A típussablonnal paraméterezett osztály **(generic class)**, lehetővé teszi, hogy más osztályok definiálásához felhasználjuk a paraméterezett osztályt, ezáltal egy adott osztálydefiníció minden típus esetén alkalmazható lesz. Tehát az osztály példányosítása során átadjuk azt a típust, amellyel szeretnénk használni az osztályunkat, majd ha más típussal szeretnénk tovább dolgozni, akkor nincs más teendő, mint új objektumpéldány létrehozása és a megfelelő típus átadása.

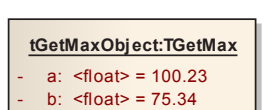
Írjunk olyan osztálysablont, amely két tetszőleges típusú adatról képes eldönteni, hogy melyik a nagyobb. (CA_08)

Mintapélda

Osztálydiagram (Class diagram)



Objektumdiagram (Object diagram)



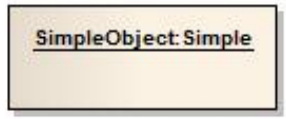
Az osztály deklarációját az *(TGetMax.h)* állományban adjuk meg. A szükséges paramétereket az osztályok és a tagfüggvények elé kell beszúrni, valamint a paraméterek típusát is jelölni kell. A konstruktorunk paraméteres, ahol két darab T-típusú adatot vár paraméterül. A ***GetMax()*** függvény egy T-típusú adatot ad vissza a destruktor pedig a memória felszabadításért lesz felelős. A *(TGetMax.cpp)* állomány tartalmazza az osztály tagfüggvényeinek a kidolgozását. A tagfüggvény neve előtt a :: hatókör-operátort használjuk arra, hogy jelezzük melyik osztály tagfüggvényét szeretnénk kifejteni. A *(main.cpp)* állomány tartalmazza a **main()** függvényt. A mainben történő példányosítás után hívjuk meg a szükséges függvényeket. A példányosítás során átadjuk a típust, amellyel az osztályt szeretnénk használni.

Gyakorló példák

Írjon egy Simple osztályt, melynek egy konstruktora van és kiírja, amikor meghívják. A main() függvényben deklaráljunk egy Simple-objektumot. A deklarált objektumot egészítsük ki egy destruktorral, mely szintén csak egy üzenetet ír ki. (GYF_CA_05)

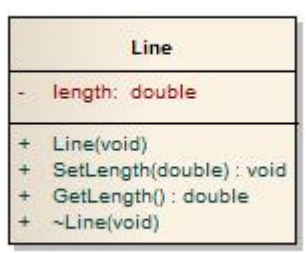
Osztálydiagram (Class diagram)

Objektumdiagram (Object diagram)

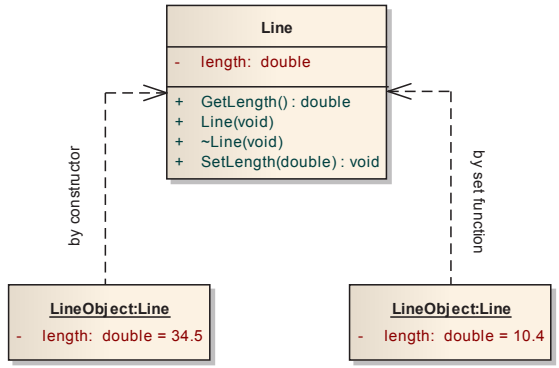


Írjon egy Vonal osztályt, melynek egy konstruktora van és kiírja, hogy „Object is being created," **illetve beállítja a vonal hosszát 34.6 cm-re. Az osztálynak legyen egy SetLength() függvénye, amely paraméterül kap egy hossz- értéket és azt állítja be a vonal hosszának. Továbbá létezzen egy GetLength() függvény, ami visszaadja a vonal hosszát! A vonal adattag típusa legyen** double**, a láthatósága az egységbezárás elvét figyelembe tartva,** private**. Dinamikusan kezeljük a memóriát, tehát hozzunk létre egy objektumot kézzel és egy destruktorral szüntessük meg. A destruktort megvalósító függvény pedig írja ki, hogy „Object is being deleted." A vonal hosszát egy konzolos kiíratással szemléltessük. (GYF_CA_06)**

Osztálydiagram (Class diagram)

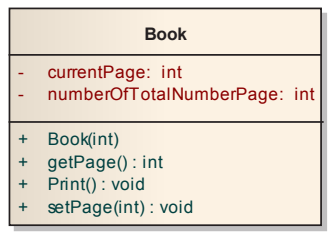


Osztály és – objektumdiagram (Object diagram)

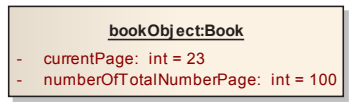


Írjon egy Book osztályt, amely egy könyvhasználatot modellez. A konstruktor segítségével állítsuk be a maximális oldalszámot. A setPage() függvény segítségével adjuk meg, hogy hányadik oldalt olvassuk. A getPage() függvény segítségével pedig írjuk ki a konzolra az adatokat. (GYF_CA_07)

Osztálydiagram (Class diagram)

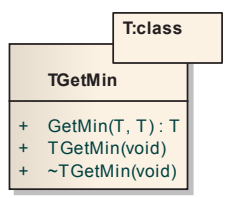


Objektumdiagram (Object diagram)

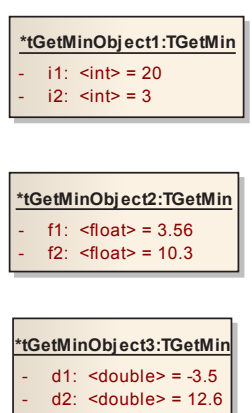


Írjon olyan minimum függvényt, amelyek két egész, valós és lebegőpontos változó közül kiválasztja, és annak értékét adja vissza! (GYF_CA_08)

Osztálydiagram (Class diagram)

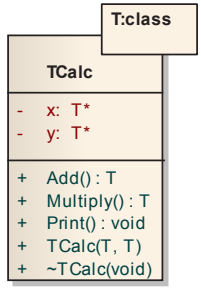


Objektumdiagram (Object diagram)

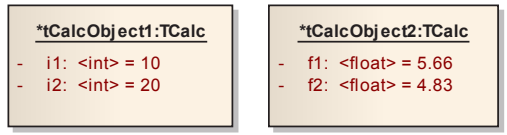


Írjon olyan osztálysablont, ahol a konstruktor bármilyen típusú változót paraméterül vár és értékül adja az osztály adattagjainak. Az osztály adattagjain végezze el a következő műveleteket: szorzás, összeadás. Használjon adattag-mutatókat! (GYF_CA_09)

Osztálydiagram (Class diagram)



Objektumdiagram (Object diagram)



2. Forráskódok

Művelet osztály deklarációja (Muvelet.h)

```
/*****************************************************************
* Muvelet osztály deklarációja                                   *
* Demonstrációs célok:                                           *
*                -> alapértelmezett konstruktor                  *
*                -> paraméteres konstruktor                      *
*                -> másoló konstruktor                           *
*                -> objektum paraméter                           *
*                -> inline módszer                               *
*                                                                *
* Katona József     <katonaj@mail.duf.hu>                        *
*****************************************************************
/
#ifndef MUVELET_H //if not define egy szimbólum nemlétét ellenőrzi
#define MUVELET_H //egy szimbólum definiálása és értékének megadása
//===============================================================
class Muvelet {
//===============================================================
private:
      double x, y, eredmeny;
      char muv;

public:
      //alapértelmezett konstruktor (inline módszer)
      Muvelet() {x=1; y=1; muv='+';}
      //paraméteres konstruktor
      Muvelet(double x1, double y1, char muv1);
      //másoló konstruktor
      Muvelet(const Muvelet &v);
      void Szamol();
      void Kiir();
};
#endif //feltétel lezárása
```

Művelet osztály implementálása (Muvelet.cpp)

```
/*****************************************************************
* Muvelet osztály definiálása/implementálása                     *
* Demonstrációs célok:                                           *
*                -> alapértelmezett konstruktor                  *
*                -> paraméteres konstruktor                      *
*                -> másoló konstruktor                           *
*                -> objektum paraméter                           *
*                -> inline módszer                               *

*                -> std névtér                                   *
*                                                                *
* Katona József     <katonaj@mail.duf.hu>                        *
*****************************************************************/
#include <iostream>           //C++ alapvető adatfolyam I/O rutinok
#include "Muvelet.h"          //a Muvelet osztály deklarációi

using namespace std;          //az std névtér behívása

//---------------------------------------------------------------
Muvelet :: Muvelet(double x1, double y1, char muv1)
//---------------------------------------------------------------
{
        this->x=x1; this->y=y1; this->muv=muv1;
}

//---------------------------------------------------------------
Muvelet :: Muvelet(const Muvelet &v)
//---------------------------------------------------------------
{
        this->x=v.x; this->y=v.y; this->muv=v.muv;
}

//---------------------------------------------------------------
void Muvelet :: Szamol()
//---------------------------------------------------------------
{
        switch(muv)
        {
         case '+': eredmeny = x+y; break;
         case '-': eredmeny = x-y; break;
         case '*': eredmeny = x*y; break;
         case '/': eredmeny = x/y; break;
        }
}
```

```
//-----------------------------------------------------------------
void Muvelet :: Kiir()
//-----------------------------------------------------------------
{
    cout << x << " " << muv << " " << y
         << " = " << eredmeny << endl;
}
```

Művelet osztály példányosítása (main.cpp)

```
/*****************************************************************
* Muvelet osztály példányosítása                                 *
* Demonstrációs célok:                                           *

*                -> különböző típusú konstruktorok hívások       *
*                -> strchr()                                     *
*                -> cin függvény                                 *
*                                                                *
* Katona József     <katonaj@mail.duf.hu>                        *
*****************************************************************/
#include <iostream>           //C++ alapvető adatfolyam I/O rutinok
#include <cstring>            //C++ verziója, C "string.h"-nak
#include <conio.h>            //DOS konzol I/O rutinok hívása
#include "Muvelet.h"          //a Muvelet osztály deklarációi

using namespace std;          //az std névtér behívása

//*****************************************************************
int main()
//*****************************************************************
{
    double a1, b1;
    char m;
    Muvelet w1;                  //alapértelmezett konstruktor hívása
    w1.Szamol();
    w1.Kiir();
    cout << "1.adat: "; cin >> a1;
    cout << "2.adat: "; cin >> b1;

    do
```

```
    {
        cout << "muveleti jel (+,-,*,/): "; cin >> m;
    } while(!strchr("+-*/",m));

    Muvelet w2(a1, b1, m);       //a paraméteres konstruktor hívása
    w2.Szamol();
    w2.Kiir();
    Muvelet w3(w2);              //a másoló konstruktor hívása
    w3.Szamol();
    w3.Kiir();

    getch();
    return 0;
}
```

Rectangle osztály deklarációja (Rectangle.h)

```
/*****************************************************************
* Rectangle osztály deklarációja                                 *
* Demonstrációs célok:                                           *
*                -> adattag mutatók                              *
*                -> dinamikus memória kezelés OOP-ben            *
*                -> destruktor                                   *

*                -> inline módszer                               *
*                                                                *
* Katona József     <katonaj@mail.duf.hu>                        *
*****************************************************************/
#ifndef RECTANGLE_H //if not define egy szimbólum nemlétét ellenőrzi
#define RECTANGLE_H //egy szimbólum definiálása és értékének megadása
//===============================================================
#include <iostream>           //C++ alapvető adatfolyam I/O rutinok

using namespace std;          //az std névtér behívása

//===============================================================
class Rectangle {
//===============================================================
private:
        double *a, *b;

public:

        Rectangle(double magassag, double szelesseg)
```

```
        {
                //dinamikus helyfoglalás
                //az adattag-mutatók számára
                this->a = new double(magassag);
                this->b = new double(szelesseg);
        }

        double Area()
        {
                return 2*(*a+*b);
        }

        double District()
        {
                return *a**b;
        }

        void Printer()
        {
                cout << "A TEGLALAP ADATAI: " << endl;
                cout << "================== " << endl;
                cout << "a oldal = " << *a << " cm" << endl;
                cout << "b oldal = " << *b << " cm" << endl;
                printf("Kerulet: %.2lf cm", this->Area());
                cout << endl;
                printf("Terulet: %.2lf cm^2", this->District());
        }

        ~Rectangle()
        {

                delete a;  //adattagmutató törlése a memóriából
                delete b;  //adattagmutató törlése a memóriából
        }
};
#endif //feltétel lezárása
```

Rectangle osztály példányosítása (main.cpp)

```
/*****************************************************************
* Rectangle osztály példányosítása                               *
* Demonstrációs célok:                                           *
*                -> dinamikus helyfoglalású objketumpéldány      *
*                -> "->" operátor                                *
*                                                                *
* Katona József     <katonaj@mail.duf.hu>                        *
*****************************************************************/
#include <iostream>           //C++ alapvető adatfolyam I/O rutinok
#include <conio.h>            //DOS konzol I/O rutinok hívása
#include "Rectangle.h"        //a Rectangle osztály deklarációi

//*****************************************************************
int main()
//*****************************************************************
{
        //dinamikus helyfoglalású objektumpéldány
        Rectangle *RectangleObject = new Rectangle(10.2, 21.5);
        RectangleObject->Area();
        RectangleObject->District();
        RectangleObject->Printer();
        //objektumpéldány meszüntetése
        delete RectangleObject;

        getch();
        return 0;
}
```

Swap osztály deklarációja (Swap.h)

```
/*****************************************************************
* Swap osztály deklarációja                                      *
* Demonstrációs célok:                                           *
*                -> memóriacím szerinti paraméterátadadás        *
*                                                                *
* Katona József     <katonaj@mail.duf.hu>                        *
*****************************************************************/
#ifndef SWAP_H //if not define egy szimbólum nemlétét ellenőrzi
#define SWAP_H //egy szimbólum definiálása és értékének megadása
```

```
//=================================================================
class Swap {
//=================================================================
        public:
                Swap(void);
                //függvény értékátadás cím szerint
                void IntegerSwapper(int& v1, int& v2);
                void FloatSwapper(float& v1, float& v2);
                void CharSwapper(char& v1, char& v2);
                ~Swap(void);
};
#endif //feltétel lezárása
```

Swap osztály implementálása (Swap.cpp)

```
/*****************************************************************
* Swap osztály definiálása/implementálása                        *
* Demonstrációs célok:                                           *
*                -> különböző típusú adatok cseréje               *
*                                                                 *
* Katona József     <katonaj@mail.duf.hu>                         *
*****************************************************************/
#include "Swap.h"                        //Swap osztály deklarációi

//-----------------------------------------------------------------
Swap::Swap(void) {}
//-----------------------------------------------------------------

//-----------------------------------------------------------------
void Swap::IntegerSwapper(int& v1, int& v2)
//-----------------------------------------------------------------
{
        int temp;
        temp = v1;
        v1 = v2;
        v2 = temp;
}

//-----------------------------------------------------------------
void Swap::FloatSwapper(float& v1, float& v2)
//-----------------------------------------------------------------
{
        float temp;
        temp = v1;
```

```
        v1 = v2;
        v2 = temp;
}

//-----------------------------------------------------------------

void Swap::CharSwapper(char& v1, char& v2)
//-----------------------------------------------------------------
{
        char temp;
        temp = v1;
        v1 = v2;
        v2 = temp;
}

//-----------------------------------------------------------------
Swap::~Swap(void) {}
//-----------------------------------------------------------------
```

Swap osztály példányosítása (main.cpp)

```
/*****************************************************************
* Swap osztály példányosítása                                    *
* Demonstrációs célok:                                           *
*                -> Swap osztály példányosítása                  *
*                                                                *
* Katona József     <katonaj@mail.duf.hu>                        *
*****************************************************************/
#include <iostream>           //C++ alapvető adatfolyam I/O rutinok
#include <conio.h>            //DOS konzol I/O rutinok hívása
#include "Swap.h"             //a Swap osztály deklarációi

using namespace std;          //az std névtér behívása

//*****************************************************************
int main()
//*****************************************************************
{
        int   i1=10,  i2=-4;
        float f1=1.5, f2=2.8;
        char  c1='*', c2='#';
```

```
        Swap *objectOfSwap = new Swap();
        cout << "csere elott  i1 : " << i1 << " i2 : " << i2 << endl;
        objectOfSwap->IntegerSwapper(i1, i2);
        cout << "csere utan   i1 : " << i1 << " i2 : " << i2 << endl;
        cout << "csere elott  f1 : " << f1 << " f2 : " << f2 << endl;
        objectOfSwap->FloatSwapper(f1, f2);
        cout << "csere utan   f1 : " << f1 << " f2 : " << f2 << endl;
        cout << "csere elott  c1 : " << c1 << " c2 : " << c2 << endl;
        objectOfSwap->CharSwapper(c1, c2);
        cout << "csere utan   c1 : " << c1 << " c2 : " << c2 << endl;
        delete objectOfSwap;
        _getch();
        return 0;
}
```

OSwap osztály deklarációja (OSwap.h)

```
/*******************************************************************
* OSwap osztály deklarációja                                       *
* Demonstrációs célok:                                             *
*                -> függvények túlterhelése/felüldefiniálása       *
*                                                                  *
* Katona József     <katonaj@mail.duf.hu>                          *
*******************************************************************/
#ifndef OSWAP_H     //if not define egy szimbólum nemlétét ellenőrzi
#define OSWAP_H     //egy szimbólum definiálása és értékének megadása
//===================================================================
class OSwap {
//===================================================================
public:
        OSwap(void);
        void Swapper(int& v1, int& v2);
        void Swapper(float& v1, float& v2);
        void Swapper(char& v1, char& v2);
        ~OSwap(void);
};
#endif //feltétel lezárása
```

OSwap osztály implementálása OSwap.cpp)

```
/*****************************************************************
* OSwap osztály definiálása/implementálása                       *
* Demonstrációs célok:                                           *
*                -> túlterhelt függvények kifejtése              *
*                                                                *
* Katona József     <katonaj@mail.duf.hu>                        *
*****************************************************************/
#include "OSwap.h"                      //OSwap osztály deklarációi

//---------------------------------------------------------------
OSwap::OSwap(void) {}
//---------------------------------------------------------------

//---------------------------------------------------------------
void OSwap::Swapper(int& v1, int& v2)
//---------------------------------------------------------------
{
        int temp;
        temp = v1;
        v1 = v2;
        v2 = temp;
}

//---------------------------------------------------------------
void OSwap::Swapper(float& v1, float& v2)
//---------------------------------------------------------------
{
        float temp;
        temp = v1;
        v1 = v2;
        v2 = temp;
}
//---------------------------------------------------------------
void OSwap::Swapper(char& v1, char& v2)
//---------------------------------------------------------------
{
        char temp;
        temp = v1;
        v1 = v2;
        v2 = temp;
}

//---------------------------------------------------------------
OSwap::~OSwap(void) {}
//---------------------------------------------------------------
```

OSwap osztály példányosítása (main.cpp)

```
/*******************************************************************
* OSwap osztály példányosítása                                     *
* Demonstrációs célok:                                             *
*                -> túlterhelt függvények hívásai                  *
*                                                                  *
* Katona József     <katonaj@mail.duf.hu>                          *
*******************************************************************/
#include <iostream>             //C++ alapvető adatfolyam I/O rutinok
#include <conio.h>              //DOS konzol I/O rutinok hívása
#include "OSwap.h"              //az OSwap osztály deklarációi

using namespace std;            //az std névtér behívása

//*******************************************************************
int main()
//*******************************************************************
{
        int   i1 = 10,  i2 = -4;
        float f1 = 1.5, f2 = 2.8;
        char  c1 = '*', c2 = '#';

        OSwap *swapperObject = new OSwap();
        cout << "csere elott  i1 : " << i1 << " i2 : " << i2 << endl;
        swapperObject->Swapper(i1, i2);
        cout << "csere utan   i1 : " << i1 << " i2 : " << i2 << endl;
        cout << "csere elott  f1 : " << f1 << " f2 : " << f2 << endl;
        swapperObject->Swapper(f1, f2);
        cout << "csere utan   f1 : " << f1 << " f2 : " << f2 << endl;
        cout << "csere elott  c1 : " << c1 << " c2 : " << c2 << endl;
        swapperObject->Swapper(c1, c2);
        cout << "csere utan   c1 : " << c1 << " c2 : " << c2 << endl;
        delete swapperObject;

        _getch();
        return 0;
}
```

TSwap osztály deklarációja (TSwap.h)

```
/*****************************************************************
* TSwap osztály deklarációja                                     *
* Demonstrációs célok:                                           *
*                -> függvénysablonok                             *
*                -> template kulcsszó                            *
*                                                                *
* Katona József     <katonaj@mail.duf.hu>                        *
*****************************************************************/
#ifndef TSWAP_H     //if not define egy szimbólum nemlétét ellenőrzi
#define TSWAP_H     //egy szimbólum definiálása és értékének megadása
//===============================================================
template <class T> class TSwap {
//===============================================================
        public:
                TSwap(void);
                void swapper(T& v1, T& v2);
                ~TSwap(void);
};
#endif //feltétel lezárása
```

TSwap osztály implementálása (TSwap.cpp)

```
/*****************************************************************
* TSwap osztály definiálása/implementálása                       *
* Demonstrációs célok:                                           *
*                -> függvénysablonok kifejtése                   *
*                                                                *
* Katona József     <katonaj@mail.duf.hu>                        *
*****************************************************************/
#include "TSwap.h"                      //TSwap osztály deklaráció

//---------------------------------------------------------------
template <class T> TSwap<T> :: TSwap(void) {}
//---------------------------------------------------------------

//---------------------------------------------------------------
template <class T> void TSwap<T> ::swapper(T& v1, T& v2)
//---------------------------------------------------------------
{
        T temp;
        temp  = v1;
        v1 = v2;
        v2 = temp;
}

//---------------------------------------------------------------
template <class T> TSwap<T> :: ~TSwap(void) {}
//---------------------------------------------------------------
```

TSwap osztály példányosítása (main.cpp)

```
/*****************************************************************
* TSwap osztály példányosítása                                   *
* Demonstrációs célok:                                           *
*                -> függvénysablonok hívása különböző típusokkal  *
*                                                                *
* Katona József     <katonaj@mail.duf.hu>                        *
*****************************************************************/
#include <iostream>          //C++ alapvető adatfolyam I/O rutinok
#include <conio.h>           //DOS konzol I/O rutinok hívása
#include "TSwap.cpp"         //a TSwap osztály deklarációi

using namespace std;         //az std névtér behívása

//****************************************************************
int main()
//****************************************************************
{
        int   i1 = 10,  i2 = -4;
        float f1 = 1.5, f2 = 2.8;
        char  c1 = '*', c2 = '#';

        TSwap<int> *p1 = new TSwap<int>();
        cout << "csere elott  i1 : " << i1 << " i2 : " << i2 << endl;
        p1->swapper(i1, i2); //függvénysablon hívása integer típussal
        cout << "csere utan   i1 : " << i1 << " i2 : " << i2 << endl;
        delete p1;

        TSwap<float> *p2 = new TSwap<float>();
        cout << "csere elott  f1 : " << f1 << " f2 : " << f2 << endl;
        p2->swapper(f1, f2); //függvénysablon hívása float típussal
        cout << "csere utan   f1 : " << f1 << " f2 : " << f2 << endl;
        delete p2;

        TSwap<char> *p3 = new TSwap<char>();
        cout << "csere elott  c1 : " << c1 << " c2 : " << c2 << endl;
        p3->swapper(c1, c2); //függvénysablon hívása char típussal
        cout << "csere utan   c1 : " << c1 << " c2 : " << c2 << endl;
        delete p3;

        _getch();
        return 0;
}
```

TGetMax osztály deklarációja (TGetMax.h)

```
/*****************************************************************
* TGetMax osztály deklarációja                                   *
* Demonstrációs célok:                                           *
*                -> osztálysablon deklaráció                     *
*                                                                *
* Katona József     <katonaj@mail.duf.hu>                        *
*****************************************************************
/
#ifndef TGETMAX_H //if not define egy szimbólum nemlétét ellenőrzi
#define TGETMAX_H //egy szimbólum definiálása és értékének megadása
//===============================================================
template <class T> class TGetMax {
//===============================================================
        T a, b;     //alapértelmezett láthatóságú T típusú adattag

     public:
        TGetMax(T first, T second);
        T GetMax();
       ~TGetMax(void);
};
#endif //feltétel lezárása
```

TGetMax osztály implementálása (TGetMax.cpp)

```
/*****************************************************************
* TGetMax osztály definiálása/implementálása                     *
* Demonstrációs célok:                                           *
*                -> függvénysablonok kifejtése                   *
*                                                                *
* Katona József     <katonaj@mail.duf.hu>                        *
*****************************************************************/
#include "TSwap.h"                        //TSwap osztály deklaráció

//---------------------------------------------------------------
template <class T> TSwap<T> :: TSwap(void) {}
//---------------------------------------------------------------

//---------------------------------------------------------------
template <class T> void TSwap<T> ::swapper(T& v1, T& v2)
//---------------------------------------------------------------
{
        T temp;
        temp  = v1;
```

```
        v1 = v2;
        v2 = temp;
}

//-----------------------------------------------------------------
template <class T> TSwap<T> :: ~TSwap(void) {}
//-----------------------------------------------------------------
```

TGetMax osztály példányosítása (main.cpp)

```
/*******************************************************************
* TGetMax osztály deklarációja                                     *
* Demonstrációs célok:                                             *
*                -> osztálysablon deklaráció                       *
*                                                                  *
* Katona József     <katonaj@mail.duf.hu>                          *
*******************************************************************
/
#ifndef TGETMAX_H //if not define egy szimbólum nemlétét ellenőrzi
#define TGETMAX_H //egy szimbólum definiálása és értékének megadása
//=================================================================
template <class T> class TGetMax {
//=================================================================
        T a, b;        //alapértelmezett láthatóságú T típusú adattag

     public:
        TGetMax(T first, T second);
        T GetMax();
       ~TGetMax(void);
};
#endif //feltétel lezárása
```

3. Felhasznált irodalom

Andrei Alexandrescu–Herb Sutter(2005): *C++ kódolási szabályok.* Kiskapu kiadó.
Robert A. Maksimchuck–Eric J. Naiburg (2006): *UML földi halandóknak.* Kiskapu kiadó.
Alex Allain (2013): *Jumping Into C++*. Cprogramming.com kiadó.
Mike McGrath(2011): *C++ Programming In Easy Steps 4th Edition*. In Easy Steps Limited kiadó.
Bjarne Stroustrup (2013): *The C++ Programming Language*. Addison Wesley kiadó
Nicolai M. Josuttis (2012): *The C++ Standard Library: A Tutorial and Reference.* Addison Wesley kiadó.
Ivor Horton (2014): *Ivor Horton's Beginning Visual C++ 2013* (Wrox Beginning Guides). John Wiley & Sons kiadó.
David Vandevoorde (2014): *C++ Templates: The Complete Guide.* Addison Wesley kiadó.
Marc Gregoire (2014): *Professional C++*. John Wiley & Sons kiadó.
Martin Fowler(2003): *UML Distilled: A Brief Guide to the Standard Object Modeling Language* (Object Technology Series). Addison Wesley kiadó.
Craig Larman(2004): *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development.* Prentice Hall kiadó.
Simon Bennet–Steve Mcrobb–Ray Farmer (2006): *Object-Oriented Systems Analysis and Design Using UML.* McGraw-Hill Higher Education kiadó.
David P. Tegarden–Alan Dennis–Barbara Haley Wixom (2012): *Systems Analysis and Design with UML.* John Wiley & Sons kiadó.

Segédanyagok
Továbbá számos olyan nagyobb magyar és külföldi egyetemektől származó publikáció, könyv és segédlet került felhasználásra, amelyek külön nem kaptak helyet a felsorolásban.