Molnár Martin
2018

A programozás alapjai 2 – I. NZH összefoglaló segédlet

blackbox kidolgozás

FIGYELEM! A kidolgozás átesett néhány javításon, de oktatói korrektúrára még nem került sor. Emiatt kérnék mindenkit, hogy kritikával kezelje az alább olvasottakat. A kidolgozás elsősorban az AUT által kiadott előadás slideok és laborsegédletek alapján készült.

REFERENCIA

Mire jók a referenciák?

A referencia egy alias, azaz egy már létező változó felcímkézése. Referencia által a változó elérhető önmagán, és a referenciáján keresztül. Egy int& y=x; referencia a már meglévő int x=20; változó új címkéje, arra a memóriaterületre mutat, ahol x van. Referenciaként átadva egy változót függvénynek lehetőségünk nyílik pointerezés nélkül a függvényen belül módosítani a változó értékét. Fontos, hogy egy függvény lokális változójával TILOS referenciaként visszatérni.

Mikor használjuk?

Referenciát használhatunk tehát a már fent említett esetben, amikor módosítani szeretnénk egy változó értékét függvényen belül. Ha egy függvény nem módosít egy változón, ám az nem beépített típusú, akkor is érdemes a lassú másolás elkerülése érdekében referenciát használni. Érdemes konstans referenciát használni, ha nem szeretnénk megváltoztatni, hogy valóban megakadályozzuk ezt.
#protipp: konstans referencia = referencia egy konstansra

Mikor kapnak kezdőértéket?

Referenciát KÖTELEZŐ inicializálni, inicializálatlan referenciát tehát nem hozhatunk létre.
Így a referenciára nem alkalmazható az értékadás, illetve magát a referenciát megváltoztatni sem tudjuk.

Mi van, ha tagváltozók?

Ha egy osztály referencia-tagváltozóval rendelkezik, akkor külön figyelmet igényel. Objektumpéldányok konstruálásánál ügyelni kell az inicializálásra. Ezt úgy oldjuk meg, hogy a referencia-tagváltozóknak mindig inicializációs listán adunk kezdeti értéket, pl. DummyClass::DummyClass(int n):ref(n){}.

FÜGGVÉNYNÉV TÚLTERHELÉS

Mi az a függvénynév túlterhelés? Mire jó?

Egy adott függvénynévhez több definíciót is létrehozhatunk. Ezen definícióknak azonban paraméterlistában is különbözniük kell, mert a C++-ban a visszatérési érték nem azonosítja a függvényt. Megegyező visszatérési érték is tartozhat túlterhelt függvénynevekhez, ha a függvényeknek más az argumentuma. Beszédes nevű és többször, különböző módon használt függvényeket túlterhelhetünk, ha azok argumentumlistája különbözik, pl. int add(int, int); vagy double add(double, double);.

Molnár Martin
2018

OOP & OSZTÁLYOK

Mik az objektumorientált alapelvek?

Encapsulation: egységbezárás - az egymáshoz tartozó adatok és a rajtuk végzett műveletek legyenek egy helyen tárolva, alkossanak egy egységet (osztályt). Specialization: specializálás / általánosítás - az úgyan úgy viselkedő, ugyanolyan tulajdonságokkal rendelkező dolgokat csoportosítjuk, ennek alapja az öröklés. Data hiding: adatrejtés - csak az adott interfacen keresztül lehessen kommunikálni az objektummal, ne lehessen inkonzisztens állapotba hozni.

Hogyan vigyáz magára az objektum?

Az objektum tagváltozói privát elérésűek, így csak tagfüggvényen keresztül lehet őket beállítani. A függvényeket pedig nekünk kell úgy létrehozni, hogy ellenőrzött és érvényes adatokat állítsanak be tagváltozók értékeinek.

Hogyan intézi az egységbezárást?

Egy adott osztályba tartoznak a tagváltozók. Ezen tagváltozókon különböző függvényeket értelmezhetünk, ezek a tagfüggvények. A tagváltozókat (private) és a tagfüggvényeket (általában public) tartalmazza együtt és foglalja egységbe az osztály (class).

Mik a hozzá tartozó kulcsszavak?

Private: privát hozzáférésűek, tehát csak az adott osztály objektumai, annak tagfüggvényei érik el. Az osztályon kívüli, (globális) függvények közvetlenül nem érik el őket. (Kivéve friend, protected...). Public: publikus hozzáférésűek, az ilyen tagváltozók eléréséhez nem kell külön függvényt hívni, osztályon kívülről is elérhetőek. Protected: egy osztály objektumának közvetlen leszármazottjai érik el + ugyan az, mint a private

A getter/setter függvények mit oldanak meg? Validálás?

A getter metódusok olyan függvények, amely privát tagváltozókat kérdeznek le általában diagnosztikai céllal, és teszik őket így úgymond kívülről elérhetővé. A setter metódusok olyan függvények, amely a privát tagváltozók beállítására szolgálnak, hogy úgymond kívülről megváltoztathatók legyenek. Ügyelnünk kell arra, hogy az osztály vigyázzon magára, azaz a setter metódusok esetében ellenőrizni kell a beállítandó adat érvényességét, validitását.

KONSTRUKTOROK

Milyen konstruktorokat ismersz? Mire jók? Hogyan állítják be a tagváltozókat? Mikor milyen hívódik meg?

Ha nem írunk egyetlen konstruktort sem, akkor létrejön egy default (0 param) konstruktor, amely nem csinál semmit. Ha azonban legalább egyet írunk, akkor nem lesz default konstruktor. Létezik egy- illetve többargumentumú konstruktor is, melyek közül az előbbinek egy speciális fajtája a copy konstruktor. A konstruktorok feladata, hogy példányokat hozzanak létre. A konstruktorok a tagváltozókat a paraméterként kapott értékek alapján állítják be általában, de kivételes esetek is vannak (const, &, static), amikor csak inicializációs listán (hamarabb meghívódik mint a törzs) adhatunk kezdeti értéket bizonyos típusú tagváltozóknak. Ha az osztálynak egy objektum tagváltozója van, akkor ha nem default konstruktorral szeretnénk, hogy létrejöjjön, vagy ha az egyáltalán nincs is neki, akkor az inicializácós listán kell kezdőértéket adnunk neki paraméterekkel. Egyébként a konstruktor törzsében nekünk kell a paraméterként kapott értékekkel felruházni a tagváltozókat. Ha nem írtunk semmilyen konstruktort, akkor paraméter nélkül a default konstruktor hívódik meg. Paramétereknek alapértelmezett értéket adva többféle variációban is meghívhatjuk a konstruktort, tehát ez is

túlterhelhető. Mindig kell lennie default, azaz paraméter nélküli konstruktornak, de ha ez alapértelmezett paraméterekkel megoldható, akkor nem kell külön írni.

Mikor történhet másoló konstruktor hívás? Mi történhet, ha nincs, vagy rosszul van megírva?

Ha egy másik példányt adunk paraméterként, akkor az alapértelmezett copy konstruktor hívódik meg, amely sekély másolatot készít. A sekély másolat azt jelenti, hogy az adott példány bitről bitre másolódik. Dinamikus adattagok esetén, ha az eredeti példány megszűnik, akkor elveszítjük a másolatunkat is, hiba történik. Emiatt kell nekünk megírni a copy konstruktort, amely különösen fontos dinamikus adattagok esetén, mert ekkor deep copy-t kell készítsünk, az előzőben leírt indokok miatt. Mindig írjunk copy konstruktort ha az osztály objektumaihoz dinamikusan lefoglalt adatterület tartozik.

DINAMIKUS ADATSZERKEZETEK

Hogyan kezelünk a heapen adatszerkezeteket? És ha tömböket hozunk létre? Hogyan szabadítjuk fel? Milyen konstruktorok hívódnak meg?

A heap dinamikus memóriaterületen foglal memóriát és szabadít fel a malloc-free páros. Ezekkel az a probléma, hogy a malloc használatakor nem hívódik meg a konstruktor, azért nekünk kellene egyesével értéket adni egy objektum tagváltozóinak. Ennek elkerülése érdekében a free storeban foglalunk memóriát a new függvénnyel. Ennek párja a delete függvény, mely segítségével visszaadhatjuk a lefoglalt területet. Tömb létrehozásakor ügyelnünk kell arra, hogy new típus [méret] szintaktikát használjunk, tehát a szükséges méret álljon rendelkezésre fordítási időben. Az ilyen jellegű memória felszabadításakor pedig a delete [] szintaktika szükséges, hogy az egész elkért terület felszabadulhasson. Ha egy osztály dinamikus adattagot használ, akkor a destruktorban szabadítsuk azt fel.

Referencia paraméterek és visszatérési értékek használata hogyan működik?

Ha egy objektumot egyszerűen adunk át paraméterként egy függvénynek, akkor másolat készül róla, és azt kapja meg a függvény. Ha egy másoló konstruktorral történik ez, akkor végtelen másoló konstruktor rekurzióba futunk. Ezt úgy kerülhetjük el, hogyha a másoló konstruktor paramétere mindig konstans referencia. A konstanság azért fontos, mert az eredeti példányt nem szeretnénk megváltoztatni. Visszatérési értékként is meghívódhat az ilyen konstruktor, de lokális változót nem lehet referencia visszatérési értékként kiszolgáltatni.

STATIKUS TAGVÁLTOZÓK, TAGFÜGGVÉNYEK

Mik azok a statikus tagváltozók? Mire jók?

A statikus tagváltozó egy osztály olyan tagváltozója, amely nem az egyes objektum sajátja, hanem minden egyes példányra megegyezik. Az ilyen statikus tagváltozó tehát nem az objektum, hanem az osztály tulajdonsága. Az értéke minden objektumra ugyan az. Különös tulajdonsága, hogy akkor is létezik, ha az osztályból még nem lett egy objektum sem példányosítva. Azokat a függvényeket, amelyek statikus tagváltozókon dolgoznak, statikus tagfüggvényeknek hívjuk. Ezek a függvények nem hívhatnak meg nem statikus tagfüggvényeket, és nem érnek el nem statikus tagváltozókat. Nem statikus tagfüggvényből azonban elérhetők a statikus tagváltozók. Ugyan így a függvényekkel... A statikus tagváltozók leggyakrabban olyankor alkalmazzuk, amikor egy osztály minden objektumának egy közös változót szeretnénk. Ilyenkor az osztályt jellemző statikus tagváltozó alkalmazandó. A statikus tagváltozót inicializációs listán kell

inicializálni új objektum konstruálásakor, definiálni és inicializálni pedig a header fileon kívül kell.

Mikor lehet őket használni?
Ha egy a pénzösszeget euróban is számontartó BankAccout osztályt csinálunk, akkor tipikusan statikus tagváltozó lesz az euró árfolyama, mert ez minden egyes objektumra (bankszámlára) megegyezik. Egyedi azonosítóval ellátni egy osztály objektumait egyszerű feladat, azonban ,,kézzel" megtenni igencsak butaság. Erre is alkalmas egy statikus tagváltozó, amelyet 0-val inicializálunk, majd minden egyes példányosításkor ebből vesszük az azonosító számértéket, majd növeljük egyel a statikus tagváltozót.

KONSTANS TAGVÁLTOZÓK, TAGFÜGGVÉNYEK

Mik a konstans változók? Miért szeretnénk őket függvényparaméterként használni?
A konstans változók olyan változók, amelyek a kezdeti érték megadása után nem változtat(hat)ják meg értéküket. Konstans változót emiatt mindig inicializálni kell, nem alkalmazható rá az értékadás operátor. Ha egy függvény a paraméterként kapott változót nem kell, hogy megváltoztassa, akkor erre figyeljünk, hogy ne is kerüljön sor. Ennek elkerülése érdekében a függvényeknek paraméterként konstans változót adjunk, ha azt nem akarjuk változtatni.

Mik a konstans tagváltozók, hogy kapnak értéket?
A konstans tagváltozók egy osztály olyan tagváltozói, melyek egy objektumon belül nem változnak. Konstans tagváltozót inicializálás után tehát nem változtathatunk meg, pontosan emiatt kell példányosításkor a konstruktor inicializációs listáján kezdőértéket adni nekik.

Mire jók a konstans tagfüggvények? Mikor muszáj használni őket?
Egy osztályon értelmezhetünk bizonyos függvényeket, amelyek nem változtatják meg az állapotát, azaz a változókat nem módosítják (tipikusan ilyenek a getter metódusok). Az ilyen függvények a konstans tagfüggvények; garantálják, hogy nem változtatják meg a tagváltozókat. Ha egy objektumunk konstans, akkor azon csak konstans tagfüggvények hívhatók. Ilyenkor tehát, ha konstans objektumunk van, akkor muszáj konstans tagfüggvényeket használunk.

FRIENDSHIP

Mi az a friend mechanizmus?
Osztályon kívüli függvényeknek megengedhetjük, hogy elérjék a private láthatóságú tagváltozókat (és függvényeket), illetve, hogy dolgozzanak rajtuk. Ehhez a függvény deklarációját be kell írni az osztályba, és elé kell írni a friend kulcsszót. A függvény definíciója ezután az osztályon kívül megírható. Egy komplett osztály is lehet friend, azonban ezt mindig konkrétan ki kell írni, a friendship nem tranzitív.

Mikor lehet csak friend függvényekkel megoldani egy feladatot?
Szinte minden feladat megoldható friend függvények nélkül csupán getter és setter metódusok használatával. Tipikus friend azonban például a << és a >> operátorok túlterhelése. Ezeket globálisan kell az iostream miatt megírnunk, és ekkor használhatjuk egy osztály I/O környezetbe illesztésekor a friend mechanizmust. Habár sérti az egységbezárás elvét, szükséges friend függvényt használni bizonyos operátorok túlterhelésekor. Például, ha egy nem beépített típust, mondjuk egy komplex számot (class Complex) szeretnénk hozzáadni jobbról egy konstans

*skalárhoz (10.0+c2), akkor globálisan, friendshipet használva kell túlterhelünk az operator+-ot. Ez azért szükséges, mert ekkor a függvény nem kapja meg a 0. *this paramétert, így a baloldali operandus is beállítható.*

OPERÁTOROK TÚLTERHELÉSE

Mire jó az operátor-túlterhelés?

*Az operátorok túlterhelése - a függvénynév túlterheléshez hasonlóan - az operátorok funkcióinak ,,bővítésére" szolgál. Az összeadás operátora például nem alkalmazható nem beépített típusokra, így az ilyen máshogy értelmezett operátorokat felüldefiniálhatjuk úgy, hogy illeszkedjenek egy adott osztályba. Tipikusan ilyenek az operátor tagfüggvények. Ezek egyik fontos tulajdonsága, hogy baloldali operandusuk a 0. *this pointer miatt az az objektumpéldány lesz, amire meghívjuk. Jobboldal operandus pedig tagfüggvényben lehet egy konstans skalár szám adott esetben, vagy akár az osztály egy másik példánya (Complex osztályban c1+10 vagy c1+c2). Ha a baloldali operandust meg szeretnénk változtatni, akkor globális függvényt kell írjunk (így valósíthatunk meg 10+c1 típusú műveleteket). Az ilyen globális függvényeknél alkalmazni kell a friendshipet. Az operátor túlterhelés biztosítja a lehetőséget műveletek összeláncolására. Az értékadás operátornál ügyelni kell a referenciával való visszatérésre, és arra, hogy ne konstruktor hívódjon meg a returnnél, hanem az adott példányt módosítjuk, így *this-t adjunk vissza.*

Hogyan érjük el, hogy tetsztőleges osztályt lehessen streamre írni, streamről beolvasni?

Osztály I/O környezetbe való illesztésekor saját kiíratő- és beolvasófüggvényeket írunk. Ezek visszatérési értéke mindig az adott stream típusú referencia (istream& vagy ostream&), baloldali operandusok szintén egy adott stream típusú referencia, míg jobb oldali operandusuk egy osztálypéldány referencia (Complex&). Ez a referencia kiíratás esetén lehet (és legyen is) konstans - hiszen nem változtatjuk - de beolvasáskor nem. Beolvasáskor elvárhatunk egy bizonyos formátumot, amit ellenőrzünk is. Formátumhiba esetén a paraméterként megadott streamre meghívjuk a clear függvényt és beállítjuk a failbitet (istream& is esetén is.clear(ios::failbit);).

Molnár Martin
2018

A programozás alapjai 2 – II. NZH összefoglaló segédlet

virtual blackbox kidolgozás

ÖRÖKLÉS

Mi az az öröklés, mire jó? Milyen új láthatósági típus használatos ennek során?

Az öröklés egy olyan kapcsolat, amely két osztály között állhat fennt. Elsődleges haszna, hogy egy osztály funkcionalitása, viselkedése újrafelhasználható lesz. Az ősosztálynak (base class) egy másik osztályt leszármazottjává (derived class) teszünk, ennek hatására a leszármazott osztály az rendelkezik az ősosztály tulajdonságaival, de ezen felül is rendelkezhet tagváltozókkal, tagfüggvényekkel, illetve az ősosztály minden nem privát tagját eléri, viszont fontos, hogy az ősosztály attribútumaiból el nem vehet!

Ennek kapcsán ismertük meg a protected láthatósági típust, mely lehetővé teszi az ilyen láthatósággal rendelkező tagváltozók leszármazottbeli elérését. Fontos megjegyezni, hogy a protected tagváltozókat más, külső osztály nem éri el. Az öröklés típusa határozza meg azt, hogy az ősosztályban adott láthatóságú tagok milyen láthatóságúak lesznek a leszármazottban. A szintaxisa nagyon egyszerű, az osztály neve után kell írni az ősosztályt az öröklés típusával: class Animal {}; …. class Dog : public Aimal {};. A leszármazott osztály konstruktorában mindig az inicializációs listán kell meghívni az ősosztály konstruktorát. Csak a közvetlen ős konstruktora hívható

Milyen funkciókat valósíthat meg az öröklés?

Az általánosítás (generalization) során van kettő vagy több osztályunk, melyek hasonlóan viselkednek. Ezeket általánosítva létrehozhatunk egy közös ősosztályt, melyben a közös tulajdonságok tagváltozók, a közös viselkedések tagfüggvények lesznek.

A specializáció (specialization) az általánosítás fordítottja. Ha van egy osztályunk, és szeretnénk olyan ehhez hasonló osztályokat csinálni, melyek ezen osztályon felüli extra tulajdonságokkal rendelkeznek, akkor specializálhatjuk az eredeti osztályunkat. Az ősosztály tagváltozóin és tagfüggvényein kívül a leszármazottakban ezek bővülhetnek.

Behelyettesíthetőségnek (polymorphism) nevezzük az öröklés azon tulajdonságát, hogy minden olyan helyen, ahol az ősosztály használható, ott használhatjuk annak leszármazottjait is, hiszen minden ősosztály-tulajdonsággal és -viselkedéssel rendelkeznek, de ez fordítva nem igaz.

VIRTUÁLIS TAGFÜGGVÉNYEK

Mikor kell virtuális tagfüggvényeket használnunk? Mik azok a tisztán virtuális tagfüggvények?

Ha egy leszármazott osztály felüldefiniálja az ősosztály viselkedését valamilyen módon (pl. függvénynév túlterhelés), akkor mindenképpen a leszármazottra specializált metódusoknak kell érvényesülnie. Ezen felül az ősosztály működését sem szabad elrontanunk. Erre adnak lehetőséget a virtuális tagfüggvények. Használatuk során egyszerűen az ősosztály tagfüggvény-deklarációja elé írjuk a virtual kulcsszót, így a függvény hívásakor mindig leszármazottnak-ősnek megfelelő fog lefutni.

Általánosításkor kerülhetünk szembe azzal a problémával, hogy adott hasonló viselkedésekhez nem tudunk egy közös viselkedést definiálni az ősosztálynak. Ebben az esetben hívhatjuk segítségül a tisztán virtuális tagfüggvényeket. Az ilyen tagfüggvények az ősosztályban virtuálisként deklarálva vannak, és a deklaráció egyenlővé van téve nullával (virtual foo()=0;), a leszármazottakban pedig a megszokott módon kell bánni velük.

Tisztán virtuális tagfüggvények következménye: absztrakt osztály. Mi az az interface?

Egy osztályt csak és csakis akkor lehet példányosítani, ha minden metódusa definiálva van. Ennek következményeképpen, ha egy osztály rendelkezik tisztán virtuális tagfüggvénnyel, akkor nem példányosítható, ekkor az osztály absztrakt osztálynak nevezzük. Fontos megjegyezni, hogy ha a konstruktornak kívülről hozzáférhetetlen láthatóságot adunk, akkor nem lesz absztrakt az osztály, hiszen osztályon belül lehet példányosítani. Ha egy osztálynak van tisztán virtuális tagfüggvénye, de azon kívül rendelkezik definiált tagfüggvényekkel, akkor attól még absztraktnak számít. Interfacenek nevezhetjük azokat az absztrakt osztályokat, amelyek csak tisztán virtuális tagfüggvényekkel rendelkeznek.

Mire hasznláhatóak az absztrakt osztályok?

Amikor több, közös őssel rendelkező ősztálypéldányt szeretnénk használni vagy kezelni, akkor az objektumokat tömbben tárolhatjuk. Heterogén kollekciónak nevezzük azt az erre használható megvalósítást, amikor egymástól különböző (specializált leszármazottak), de mégis hasonló (közös absztrakt ősosztály) elemeket tárolunk együtt. Fontos, hogy ha az egyik leszármazott dinamikus memóriakezelést valósít meg, akkor természetesen meg kell írnunk a destruktorát a memóriafelszabadítás miatt, ehhez azonban szükségünk van az absztrakt ősosztály destruktorának virtuálissá tételéhez.

TÖBBSZÖRÖS ÖRÖKLÉS

Mi az a többszörös öröklés? Mikor lehet szükségünk rá?

Tegyük fel, hogy van két osztályunk. Programunk írása közben szükségünk lesz még egy osztályra a már meglévő két osztály minden tulajdonságával és viselkedésével, esetleg ezen felüli plusz attribútumokkal. Ilyenkor a duplikáció elkerülése érdekében nem implementáljuk az új osztályban mégegyszer a már meglévő osztályokban lévő tagokat, hanem örököltetjük azokat. Ha tehát egy leszármazottnak több osztály tulajdonságait és viselkedéseit szeretnénk örököltetni, akkor többszörös öröklést alkalmazva, mindegyik osztályból leszármaztatjuk. Ekkor az összes ős tagváltozóit és tagfüggvényeit örökli a leszármazott. A szintaktika megegyezik az egyszerű öröklléssel, csupán vesszővel elválasztva egymás mögé kell írnunk az örökléseket.

Mikor kell virtuális többszörös öröklést alkalmazni? Mire kell ekkor figyelni?

Előfordulhat olyan eset, amikor egy közös ős leszármazottjaiból szeretnénk többszörösen örököltetni egy új osztályt. Ilyenkor azonban két ősosztály típusú objektumnak foglalódna hely a memóriában, ami pazarlás, értelmetlen, és fordítási hibát okozhat akkor, amikor olyan függvényt szeretnénk hívni a többszörös leszármazotthoz, ami az eredeti közös ősosztályhoz tartozik. A probléma kiküszöbölhető virtuális öröklléssel, ekkor nem jön létre a két specializált ősosztálynak külön-külön memóriaterület, hanem egyen osztozkodnak. Ilyen öröklés esetén az öröklés láthatósága és az ősosztály közé kell írni a virtual kulcsszót.

Miért használjuk, és miért ne használjuk a többszörös öröklés?

Egy többszörös öröklést is tartalmazó programot nagyon nehéz fenntartni és debuggolni. Ha egy osztály olyan több, különböző osztályból öröklődik, melyeknek van közös ősosztálya, virtuális öröklésre van szükség. Könnyen felismerhető, mivel ha felrajzoljuk az öröklési hierarchiát, egy gyémánt-szerű alakot kapunk, ezért is hívják "diamond problem"-nek, vagy "deadly diamond of death"-nek. Ez a diamond problem is sok fejfájást és nehézséget okozhat.
Érdemes viszont használni akkor, amikor előre adott osztályaink vannak, mert ilyenkor a keretrendszerbe való illesztéshez több osztályból származtatunk, ám ekkor a leszármazott úgymond egy hívható felületet, interfacet implementál. Általában az a megszokott, hogy egy osztályból öröklünk, de több interfacet implementálunk.

Molnár Martin
2018

TEMPLATE

Mi az a template? Mik a függvénytemplatek, mi a template paraméter?

A template sablont jelent, melyre akkor lehet szükségünk programozás során, amikor valamit úgy szeretnénk megvalósítani különböző esetekben, hogy a megvalósítás módja azonos. Függvényeknél például írhatunk rendező függvényt inteket, doubleöket tartalamzó tömbre. Ha megírtuk a rendezőalgoritmust, akkor a duplikáció elkerülése érdekében nem írunk egy másik típushoz tartozó ugyanilyen függvényt, alkalmazzunk inkább sablont. Ha egy template <class T> sablonunk van, akkor függvénytemplatenek nevezzük az olyan függvényeket, melyek paraméterül egy T típusparamétert kapnak. A függvényt erre a T típusra implementáljuk, használatkor pedig meghívhatjuk intre, doublere, stb. Figyelemebe kell venni, hogy olyan paramétert adjunk az ilyen függvényeknek, amire a függvényben használt metódusok, operátorok meg vannak írva.

Hogyan használhatók a függvénytemplatek? Milyen példányosítási módok léteznek?

Függvénytemplateket használhatunk egyszerű függvényként; ilyenkor a különböző függvényhívásokkor a paramétereknek hívásonkétn egyezniük kell – ezzel a típussal lesz példányosítva a függvény. Ezt a módszert nevezzük implicit példányosításnak.
Explicit módon is meg lehet azonban adni a függvény paramétereinek típusát. Erre bizonyos esetekben nincsen feltétlenül szükség, választhatóan adhatjuk meg a típust az előző implicit példányosításhoz hasonlóan annyi kiegészítéssel, hogy a típus < > között beírjuk a függvénynév és a paraméterlista közé (squareSum<int>(2,3);). Ilyenkor ez nem kötelező, viszont mivel a template függvény nem végez semmilyen konverziót, bizonyos esetekben muszáj konkrét típust megadni. Egy templatefüggvényt különböző paraméterekkel való hívását a fordító nem tudja végrehajtani, mert mindegyik paraméternek közös T típust vár. Ekkor megadhatjuk a kívánt típust explicit módon úgy, hogy fentebb leírtuk, vagy a függvény paramétereit azonos típusúakká konvertáljuk (squareSum(2.5, double(3));).

PARAMETRIZÁLT OSZTLÁLYOK

Mik azok a parametrizált osztályok? Hogyan használjuk az osztálytemplatet?

A templatek bevezetésekor leírt problémával találkozhatunk osztály szinten is, ekkor template osztályokat írhatunk. Osztálytemplate esetén az osztály tagjainak kell tudnia használni több típust, az függvénytemplateknél leírt szabályok mind érvényesek ilyenkor is. Az osztály definíciója elé is kell írnunk a sablont (template <class T>), és T típusparamétert használhatjuk az osztályon belül. Felparaméterezett osztályt úgy használunk, hogy az osztály neve után < > közé írjuk a kívánt típust, majd jöhet a változó neve (Stack<int> integers;). Miután megtörtént a paraméterezés, közönséges osztályként lesz használható,

Mit tudunk a template paraméterekről?

Template paraméternek adhatunk default értéket is, így akár argumentum nélkül is hozhatunk létre osztálytemplateből példányt. A default érték megadása template<class T = int> alakban történik.
Template paraméter lehet típus (beépített típus, felparaméterezett template osztály), típusos konstans vagy template osztály (nem felparaméterezett template osztály).

Molnár Martin
2018

Mi az a tagfüggvény template? Hogyan lehet örökölni template osztályból? Template VS öröklés? Lehet template osztályt dekomponálni?

Akár egyszerű, akár template osztály esetében egy tagfüggvénynek lehetnek külön templateparaméterei.
Ha egy template osztálynak rögzítjük a paramétereit, akkor osztály lesz, így már lehet belőle örököltetni. Tehát ha minden templateparaméter rögzítetté válik, akkor általános osztályként lehet örökölni a (már amúgy nem) template osztályból.
Ha ugyanazt a viselkedést szeretnénk leírni több típusra, használjunk template-et. Ha felüldefiniálható viselkedésre van szükségünk, használjunk öröklést.
Template osztályt nem lehet két fájlra bontani olyan módon, hogy az osztály deklarációja egy header, a definicíója pedig egy .cpp fájlban van, a fordítás sikertelen lesz. Tegyük egy fájlba az egy osztályhoz tartozó deklarációkat és definíciókat, így az include-oláskor a függvények definíciója is elérhetővé válik. Ettől még nem kell az osztályon belül, inline megírni a függvények törzsét, lehet az osztálydeklaráció után.

KONVERZIÓ

laborsegédletből jól átnézhető

KIVÉTELKEZELÉS

laborsegédletből jól átnézhető