

Programozás módszertan

Típusok

Pere László (pipas@linux.pte.hu)

PÉCSI EGYETEM TERMÉSZETTUDOMÁNYI KAR
INFORMATIKA ÉS ÁLTALÁNOS TECHNIKA TANSZÉK

MAGYAR TUDOMÁNYOS AKADÉMIA
SZÁMÍTÁSTECHNIKAI ÉS AUTOMATIZÁLÁI KUTATÓINTÉZET
ADATBÁZIS ÉS ADATKEZELÉSI OSZTÁLY

A típus

A típus

A típus absztrakció, a programban előforduló bizonyos objektumok (változók, függvények, stb.) közös tulajdonságait foglalja össze.

- Kódolás.
- Méret.
- Szerkezet.
- Jelentés.

Típusspecifikáció

A $\mathcal{T}_S = (T, I_S, \mathbb{F})$ típusspecifikáció, ahol:

- T az alaphalmaz.
- I_S specifikációs invariáns.
 $T_S = [I_S]$ elemek, amelyekhez I_S igaz értéket rendel, a típusérték-halmaz.
- \mathbb{F} műveletek specifikációi, amelyeknek állapotterében T_S szerepel.

Típusmegvalósítás

A $\mathcal{T} = (\rho, I, \mathbb{S})$ típus, ahol:

- $\rho \subseteq E^* \times T_S$ reprezentációs függvény.
A típusértékhalmaz elemeihez rendeli az elemiérték sorozatokat (egyhez akár többet).
- I típusinvariáns, meghatározza, hogy mely elemiérték sorozatok határoznak meg érvényes típusértékeket.
- \mathbb{S} típusműveletek, programrészek, amelyeknek állapotterében szerepel E^* .

E^* elemiérték sorozatok, az ábrázolásra használt részek sorozata.

Típus és specifikáció

A $\mathcal{T} = (\rho, I, \mathbb{S})$ megfelel $\mathcal{T}_S = (T, I_S, \mathbb{F})$ specifikációnak, ha:

- $\rho([I]) = T_S$ azaz minden, a típusinvariánsnak megfelelő elemiérték-sorozat a specifikációs invariánsnak megfelelő típusértéket ábrázol és a teljes típusérték-halmaz ábrázolható.
- $\forall F \in \mathbb{F} : \exists S \in \mathbb{S}$, hogy S a ρ -n keresztül megoldja F -et.

Azaz minden specifikált műveletre van olyan megvalósított program, amely ρ kódolást és dekódolást feltételezve elvégzi a műveletet.

Töbszörösség

Egy típusspecifikációnak több típusimplementáció (típus) is megfelelhet, amelyek eltérhetnek egymástól például az ábrázolásban, a típusműveletek megvalósításában, esetleg a típusinvariánsban.

Típusosság

A típusosság

A típus meghatározza az adott objektumon végrehajtható műveleteket. A programozási nyelv tervezésének egyik célja, hogy az objektumokon való műveletvégzés hibáit lehetőleg fordításkor fedezzük fel.

A típusosság szerint az objektumokhoz típust rendelünk, a fordító pedig ellenőrzi, hogy a típusok által leírt módon használjuk-e az objektumokat.

Típusosság

- Statikusan típusos a nyelv, ha az összes objektum típusa meghatározható fordításkor.
- Erősen típusos a nyelv, ha fordításkor ellenőrizhető, hogy a típus mindenütt megfelelő.
- Gyengén típusos a nyelv, ha bizonyos esetekben csak a futáskor fedezhető fel a típus helytelen használata.

A típuskonverzió

A típuskonverzió

A programban az objektum típusának megváltoztatása – amely szükséges lehet – a típuskonverzió.

Automatikus típuskonverzió esetén a programozó beavatkozása nélkül, automatikusan történik a típuskonverzió.

Explicit típuskonverzió esetén a programozó utasítására, előírása szerint történik típuskonverzió.

Reprezentáció megváltoztatása

Bizonyos esetekben a típuskonverzió során megváltozik az objektum ábrázolása. Ugyanazon objektumhoz más ábrázolást használunk a konverzió után.

Nem minden típuskonverzió változtatja meg az ábrázolást!

Bővítő/szűkítő

Ha szűkebb típusértékalmazról térünk át bővebb típusértékalmazra, bővítő jellegű típuskonverzióról beszélünk. Ez általában automatikusan elvégezhető, veszélytelen.

Ha bővebb típusértékalmazról szűkebb típusértékalmazra térünk át, szűkítő típuskonverziót végzünk. Ez adatvesztéssel is járhat.

Az értelmezés megváltoztatása

Az ilyen jellegű konverzió az ábrázoló bitsorozatot változatlanul hagyja, de annak értelmezését megváltoztatja.

Triviális, hogy az ilyen konverziók nem jelennek meg a bináris programban – nincs hatásuk –, csak a típusrendszert befolyásolják.

C

A C nyelv típuskényszerítés (*cast*) operátora egyaránt használható az értelmezés és az ábrázolás megváltoztatására.

- Statikus konverzió: csak hasonló típusok között (pl. egészek) között végez konverziót.
- Dinamikus konverzió: osztályok közt konvertál, futási időben végezve ellenőrzést.
- Konstans konverzió: változtatható/nem változtatható típusok közt végez váltást.
- Értelmező konverzió: azonos méretű típusok közt az értelmezés megváltoztatására szolgál.

Jellemző a leszármazott osztály konvertálása ősosztályra és az ellentétes irányú explicit konverzió futási idejű ellenőrzése.

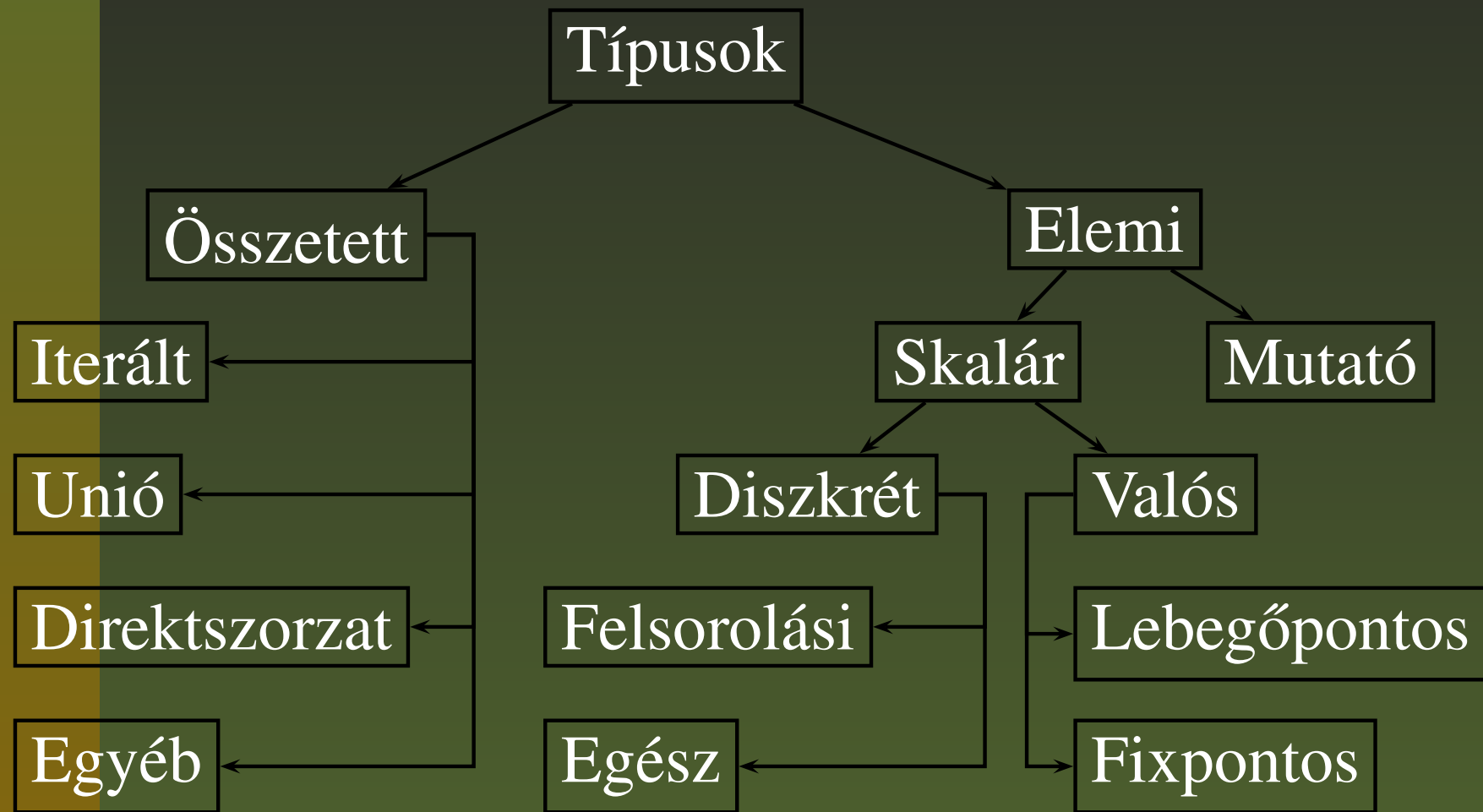
Típusok osztályozása

Típusosztályok

A következő oldalon a típusok egy vázlatos osztályozását láthatjuk. Az osztályozás alapja a típusok szerkezete.

Ez az osztályozás az egyes programozási nyelvekben is tükröződhet.

Típusosztályok



Elemi/összetett

Az elemi típusok logikailag felbonthatatlanok, belső szerkezetük egységes. Az elemi típusokat a programozási nyelv biztosítja számunkra.

Az összetett típusok az elemi típusokból és az összetett típusokból épített típusok. Ezeket a típusokat sokszor a programozó hozza létre az adott feladatnak megfelelő szerkezettel.

Skalár és mutató típusok

A skalár típusok egyetlen mennyiséget tartalmaznak (számot).

A mutató típusok memóriacímeket tárolnak, a memóriában található elemeket jelölik ki. (Bizonyos szempontból a mutató típusokat összetett típusoknak tekinthetjük.)

Valós/diszkrét típusok

A valós típusok a valós számok ábrázolására szolgálnak (a típusspecifikációban az alaphalmaz a való számok halmaza). A valós típusok esetén lebegőpontos vagy fixpontos számábrázolást használunk.

Hasonlóképp épülnek az egész számokra az egész típusok. A felsorolási típusokat általában a programozó hozza létre valamely egész típus értékalmazának szűkítésével.

A skalár típusosztály

A skalár típusosztály

A skalár típusosztály elemei általában olyan típusok, amelyeket nem a programozási nyelv, hanem a processzor hoz létre.

A mai processzorok általában támogatják az egész típusok (kettes komplementes) és a valós típusok (lebegőpontos) használatát.

Műveletek

A skalár típusok rendezettek, ezért a következő műveletek általában értelmezettek:

- Relációs operátorok: $<$, $>$, \leq , \geq , $=$, \neq .
- Rákövetkező és megelőző elem (inkrementálás és dekrementálás).
- A legnagyobb illetve legkisebb elem.

Ada

Az Ada a skalár típusokon értelmezi a következő műveleteket:

- Relációs operátorok: $<$, $<=$, $>$, $>=$, $=$, \neq , $/=$.
- Intervallum létrehozása: `range alsó..felső`.
- Az intervallumhoz tartozás megállapítása: `in`,
`not in`

Ada skalár attribútumok

- S'First, S'Last.
- S'Range a típusértékhalma, azaz `range S'First..S'Last`.
- S'Base: a típus alaptípusa.
- S'Min és S'Max: két elem közül a kisebb, illetve nagyobb.
- S'Succ és S'Pred: rákövetkező és megelőző elem.
- S'Image és S'Wide_Image: karakterlánccá alakítás (ASCII és ISO 10646).

Ada skalár attribútumok

- S'Value és S'Wide_Value: karakterlánc visszaalakítása.
- S'Width és S'Wide_Width: milyen hosszú karakterlánc lesz az adott értékből.

(a) Diszkrét típusok

A diszkrét típusosztály

A diszkrét típusosztály általában nem vezet be újabb műveleteket, de sokszor ez a legbővebb típus, amely:

- Tömbök indexelésére alkalmas.
- Előre kiszámított ciklus szervezésére használható.
- Többszörös elágazás szelektora lehet.

Ada diszkrét típusai

Az Ada a következő attribútumokat vezeti be a diszkrét típusosztályra:

- S'Pos: S-ről képez egész típusra, nyilvánvalóan a felsorolási típusok esetében hasznos.
- S'Val: egész típusról képez az adott típusra, szintén felsorolási típusok esetén használjuk.

Felsorolási típusok

A felsorolási típusok típuskonstrukciós eszközzel hoznak létre újabb típust az elemek egyszerű felsorolásával.

A felsorolási típusok ábrázolása általában egyszerűen az egész számokra való leképezéssel történik (rendezett módon).

Pascal

A Pascal nyelvcsaládban a felsorolási típuskonstrukció a következő mintára végezhető el:

```
type Weekend = (Szombat, Vasarnap);
```

C

A C nyelvcsaládban a felsorolási típuskonstrukció a következő formában végezhető el:

```
typedef enum    Kedd, Szerda, Csutortok
```

Ebben a nyelvcsaládban azonban a felsorolási típusok valójában egész típusok.

C példa

elmelet/C/01-fels.c

```
1  #include <stdio.h>
2
3  typedef enum {
4      hetfo, kedd, szerda
5  } napok;
6
7  typedef enum {
8      januar, februar, marcius
9  } honapok;
10
11 main() {
12     napok a = szerda;
13     honapok b = februar;
14     printf("%d\n", a + b);
15 }
```

Az egész típusosztály

Az egész típusosztály kezelése során a legtöbb programozási nyelv a hatékonyságra törekszik, ezért a legtöbb programozási nyelv a processzor adta egész típusokat igyekszik használni.

Fontos, hogy programjainkban törekedjünk az egészek használatára, hiszen az egész típusosztály elemeivel gyors programok készíthetők.

Típusértékhalmoz

Az egész típusosztályban általában többféle, típusértékhalmozában különböző típus is a rendelkezésünkre áll, melyek:

- Különböznek a típusértékhalmoz és a helyfoglalás méretében (8, 16, 32, 64 bites egészek).
- Az előjel kezelésében (kettes komplement, pure binary).

Műveletek

A leggyakrabban implementált műveletek:

- Egyszerű matematikai műveletek: összeadás, kivonás, szorzás osztás esetleg hatványozás.
- Bitműveletek: léptetés jobbra és balra, bitenkénti és, bitenkénti vagy, bitenkénti negáció.

Kilépés az alaphalmazból

A következő két művelet kivezethet a használt alaphalmazból akkor is, ha mindkét operandus egész:

- Osztás: sok nyelv egészeken egészosztást végez, ahol az eredmény mindig egész lesz.
- Negatív kitevőjű hatványozás: sok nyelv nem engedi a negatív kitevő használatát.

Kilépés a típusértékhalmazból

Minden egészművelet kivezethet a típusértékhalmazból. Erre a problémára a programozási nyelvek általában a következő válaszokat adják:

- Futási idejű ellenőrzést végeznek és kivételt vagy programmegszakítást kezdeményeznek.
- Nem ellenőrzik a túlcsordulást és modulo 2^n maradékosztályon végeznek műveleteket. (Kettes komplementes esetén ez különös világot vázol fel.)

Pascal

A Pascal típuskészlete megvalósításfüggő. A Turbo Pascal 8 bites ShortInt/Byte, 16 bites Integer/Word, és a 32 bites LongInt.

A nyelv választhatóan támogatja a futási idejű intervallum ellenőrzést (binárisba fordítható).

Pascal

A Pascal műveletei:

- Összeadás, kivonás, szorzás.
- `A /` kivezet az alaphalmazból, a `div` és `mod` nem.
- Bitműveletek: `shl`, `shr`, `not`, `and`, `or`, `xor`.
- Abszolútérték (`Abs`) és négyzetre emelés (`Sqr`).

Az Ada egész típusai két alosztályra oszthatók:

- Előjeles egészek: ezen típusok típusértékalmaza tartalmazhat negatív egész számokat is. Ha egy művelet eredménye nincs a típusértékalmazban, kivétel váltódik ki.
- Moduló típusok: pozitív egész számok moduló maradékosztályként való kezelése.

C

A C és C++ nyelvek szabványai nem határozzák meg, hogy az egyes egész típusok típusértékhalmaza mekkora, csak alsó határt adnak meg.

A C nyelvben túlcsordulás ellenőrzés nincs, az osztás egész osztás, a hatványozásnak pedig nincsen operátora.

Karakter és logikai típusok

Nem tárgyaljuk részletesen a diszkrét típusosztály karakter és logikai osztályait. Ezekről a kötelező irodalomból olvashatunk.

(b) Valós típusok

A valós típusok

A valós típusosztály azoknak a típusoknak a gyűjtőneve, ahol az alaphalmaz a valós számok halmaza. Ennek a halmaznak a számossága meghaladja az egész számok halmazának számosságát és ez problémákat okoz.

Tetszőlegesen kicsiny intervallumban végtelenül sok valós szám van, így a típusérték-halmaz tetszőlegesen kicsiny intervallumot sem fedhet le (és ezt hívjuk számítógépnek!).

A valós típusok csak az ábrázolásban térnek el egymástól, együtt tárgyaljuk őket.

Fixpontos számábrázolás

- Előnye az állandó pontosság.
- Hátránya, hogy nem tudjuk egyszerre ábrázolni az abszolút értékben kicsiny és nagy számokat.

Tudományos és mérnöki feladatokra nem alkalmas, pénzügyi rendszerek készítésére kiváló, ezért általában csak adatbáziskezelő nyelvek támogatják.

Lebegőpontos számábrázolás

- A pontosság a karakterisztikának megfelelő változik és ez kellemetlen.
- Egyszerre tudjuk ábrázolni a kis- és nagy abszolút értékű számokat.

A legtöbb processzor és a legtöbb programozási nyelv támogatja az IEEE 754 (1985) és az IEC 60559 (1989) alapján.

Műveletek

A leggyakrabban implementált műveletek:

- Alapműveletek és hatványozás valamint négyzetgyök.
- Szabványos könyvtári függvényként általában megvalósulnak a természetes alapú – esetleg kettes és tízes alapú – logaritmus, természetes alapú exponenciális függvény, trigonometrikus és inverz trigonometrikus függvények valamint a véletlenszám generátor.

Pascal

A Pascal lebegőpontos típusainak legtöbbje implementációfüggő. A valós típusok műveletei:

- Alapműveletek.
- Abszolútérték és egészrész, négyzetre emelés és négyzetgyök.
- Természetes alapú exponenciális és logaritmus függvény.
- Trigonometrikus függvények.
- Véletlenszám-generátor.
- Konverziós függvények.

Ada

Az Ada támogatja a lebegőpontos és a fixpontos számábrázolást is. Tetszőleges számú valós típust hozhatunk létre a pontosság megadásával.

A támogatott műveletek a négy alpművelet, az egész kitevős hatványozás és az abszolútérték.

Mutató típusok

Mutató típusok

A mutató és referencia típusok a memóriacímek absztrakciói.

- A mutatók memóriacímek tárolására alkalmas típusok.
- A referencia típusok olyan mutatók, amelyek vagy létező objektumot jelölnek ki a memóriában, vagy egy kitüntetett – sehova sem mutató – értéket hordoznak.

A következőkben általában egységesen, mutató megnevezéssel tárgyaljuk a két típusosztályt.

Típusértékhalmoz

A mutatókat általában előjel nélküli egész számként ábrázoljuk.

Bizonyos nyelvek többféle típusértékhalmozzal rendelkező mutató típust is használnak (távoli és közeli mutató), amelyek teljes címtartományban és szegmensben belüli címzésre használhatók.

Ezt a megoldást a processzorok indexelt címzési rendszere teheti indokolttá.

Típusos mutatók

Sok nyelv a mutatóhoz típusként hozzárendeli annak az objektumnak a típusát, amelynek címét a mutató hordozza.

Ez a típus a mutató ábrázolását nem befolyásolja, de lehetőség ad a mutatók használatának ellenőrzésére.

Az ilyen esetekben általában léteznek típus nélküli mutatók, amelyekre a dereferencia nem megengedett, de bennük mutató érték tárolható.

Érvénytelen érték

A nyelvekben, amelyekben mutatótípus létezik, rendelkezésünkre áll a mutató inicializálatlan állapotát jelölő érték. Azon mutatók, amelyeknek ez az értékük biztosan nem hordozzák érvényes objektum címét.

Nem hozhatunk létre olyan objektumot, amelynek a címe az inicializálatlan állapotot jelző érték.

Műveletek

A következő oldalakon a mutató típusokkal kapcsolatban használható műveleteket tárgyaljuk.

Értékadás

A mutatók értékadása általában csak memóriacím lemásolását jelenti, de szemétgyűjtéses memóriagazdálkodás esetén pl. a hivatkozott objektum hivatkozásszámlálójának növelését – és a régi érték által jelölt terület hivatkozásszámlálójának csökkentését – is jelenti.

A mutató értékül kaphatja:

- Sehová sem mutató értéket.
- Másik mutató értékét.
- Lefoglalt memóriaterület címét.
- Statikus vagy automatikus változó címét.

Memóriafoglálás

Szinte minden nyelv biztosít eszközt dinamikus memóriafoglálásra, hiszen ez nagymértékben növeli az elkészült program rugalmasságát.

Bizonyos nyelvek – pl. az objektumorientált nyelvek – elvégzik a dinamikusan foglalt memóriaterület inicializálását is a memóriaterületen tárolni kívánt objektum típusának megfelelően (típusinvariáns).

Memória-felszabadítás

Nem minden nyelv biztosít deallokátor eszközt a programozó számára. Egyes nyelvek csak a garbage collector számára teszik lehetővé a foglalt memóriaterületek felszabadítását.

Referenciaképzés

A referenciaképzés a statikus és automatikus változók címének elhelyezése mutatóban.

Bizonyos programozási nyelvek ezt nem teszik lehetővé. Ezekben a nyelvekben csak a dinamikusan foglalt memóriaterületek címét kezelhetjük mutatóval.

Mutató követése

A mutató követése, hivatkozásfeloldás (*pointer dereference*) művelet során a mutatóval megcímezzük a memóriát és a mutató által kijelölt objektumot kezeljük.

Relációk

Mutatók kapcsán általában rendelkezésünkre áll néhány relációs operátor:

- Egyenlőségvizsgálat: a legtöbb nyelvben rendelkezésünkre áll az egyenlőségvizsgálat. Két mutató egyenlő, ha pontosan ugyanazt az objektumot jelölik a memóriában.
- Kisebb, nagyobb: Bizonyos nyelvek lehetővé teszik a mutatók nagyságának összehasonlítását. A C programozási nyelv például tömbök címének összehasonlításakor garantálja, hogy a nagyobb indexhez nagyobb cím tartozik.

A C mutatóaritmetikája

A C mutatói

A C programozási nyelv – és egyes leszármazottai – igen fejlett módon támogatják a mutatók használatát és erre a C nyelven megírt programok általában kifejezetten építenek.

Ez az oka annak, hogy a C mutatóit külön tárgyaljuk (és persze az, hogy a mutatók használata nem olyan magától értetődő mint pl. az egész számoké).

A mutató típusa

A C fordító ellenőrzi az objektum és az azt jelölő mutató típusának megfeleltetését. Minden mutatóhoz típust rendel és minden értékadás során ellenőrzi a típust.

A mutató típusa – az, hogy milyen objektum címének hordozására készült a mutató – a mutatóaritmetika alapja.

A void * mutató

A void típus mérete 0, a void típust memóriában jelölő mutató a típus nélküli mutató.

Minden mutatótípus automatikusan konvertálódik a void * típusra és a void * típus automatikusan konvertálódik minden típusra.

Típus mérete

Bármely objektum vagy típus mérete megállapítható a `sizeof` prefix operátor segítségével.

Az `xxx` típus mérete (bájtban) meghatározható a `sizeof(xxx)` operátorral.

Mutatóaritmetika

Legyen p egy t típusú objektumot jelölő mutató és legyen i egy tetszőleges egész típusú objektum. Akkor a

$$p + i$$

értéke a p értékét

$$i \times \text{sizeof}(t)$$

értékkel növeli.

Az eredményül kapott új mutató annak a t típusú, legalább $i + 1$ elemet tartalmazó v vektornak a v_i elemét jelöli, amelynek v_0 eleme a p címen található.

Mutatóaritmetika

Ha p és p' t típusú mutatók és p a v_i , p' pedig ugyanazon t típusú vektor v_j elemét jelölik, akkor

$$p - p'$$

előjeles egész kifejezés értéke megegyezik

$$i - j$$

kifejezés értékével.

Következmény: $p \rho p' \Leftrightarrow i \rho j$.

Kifejezések

Függvények és operátorok

A függvények és az operátorok valójában névvel ellátott programrészletek.

A programozási nyelvek általában megengedik függvények létrehozását, de viszonylag kevés olyan programozási nyelv van, amely operátorok létrehozását vagy túlterhelését (*operator overloading*) megengedik.

A kifejezésekben viszont általában nem szükséges megkülönböztetnünk függvényeket és operátorokat.

Az aritás

A függvények és operátorok aritása az argumentumaik számát jelzi.

- Az egy argumentumú (1 aritású) függvényt és operátort *unáris*nek nevezzük.
- A kettő argumentumú (2 aritású) függvényt és operátort *bináris*nek nevezzük.
- Az általános esetben k -áris vagy k aritású függvényről vagy operátorról beszélünk.
- Bizonyos esetekben az állandókat és a változókat, valamint a 0 argumentumú függvényeket *nulláris*-nak vagy 0 aritásúnak nevezzük.

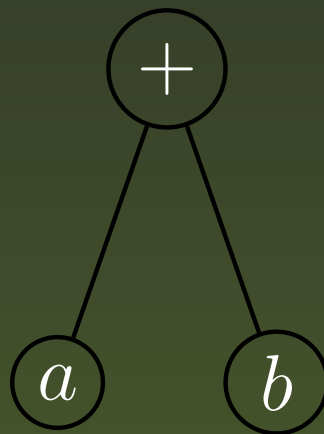
A kifejezésfa

A fák és a nyelvtanok közt fennálló intim viszony, valamint a tény, hogy a kifejezések szerkezetét nyelvtanok segítségével tudjuk kezelni mindenképpen indokolja, hogy a kifejezések szerkezetét fákkal ábrázoljuk.

- A nulláris kifejezés kifejezésfája egy csúcsból áll, amelynek címkéje a függvény.
- A k -áris függvény kifejezésfájának gyökere a függvénnyel címkézett, élek indulnak belőle az argumentumok részfáihoz.

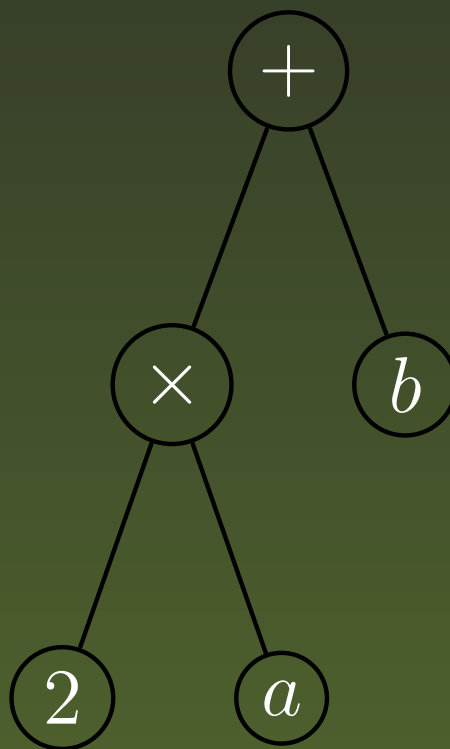
Kifejezésfa (példa)

$$a + b$$



Kifejezésfa (példa)

$$2a + b$$



Kifejezésfa kiértékelése

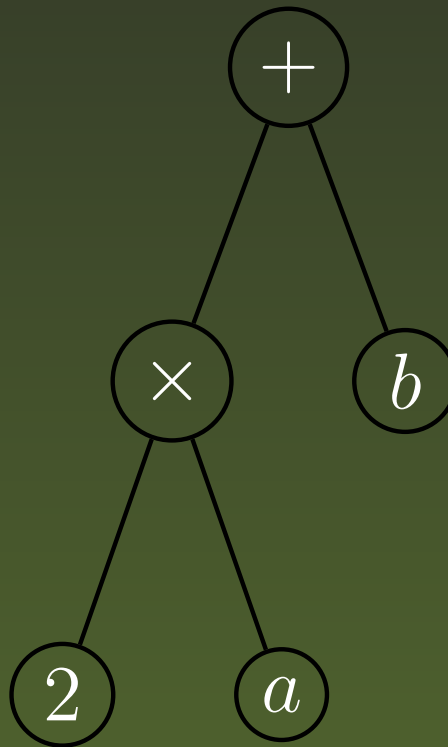
Ha egy kifejezésfa által képviselt kifejezés értékére van szükségünk, a kifejezésfát kiértékeljük. Ezt a következőképpen tehetjük meg:

- A levél értéke a változó vagy konstans értéke.
- Operátort vagy függvényt reprezentáló csomópont értéke akkor értékelhető ki, ha a leszármazottainak kiértékelése befejeződött.

A fa – vagy részfa – értéke természetesen a gyökérelem értéke.

Kiértékelés (példa)

Legyen $a = 21$ és legyen $b = 0$! Értékeljük ki a következő kifejezésfát!



Prefix operátorok

A prefix operátorokat mindig az argumentumaik elé írjuk (vö. *pre-*), azaz a prefix operátorok használatát leíró nyelvtani szabályokban mindig az operátor neve áll elől.

Az ilyen operátorokat használó nyelvtanok által generált kifejezések kifejezésfája mindig egyértelműen és egyszerűen felírható a kifejezés balról jobbra történő olvasásával.

Prefix operátorok (példa)

Legyen G nyelvtan részlete a következő:

$$K \rightarrow OEE \quad (1)$$

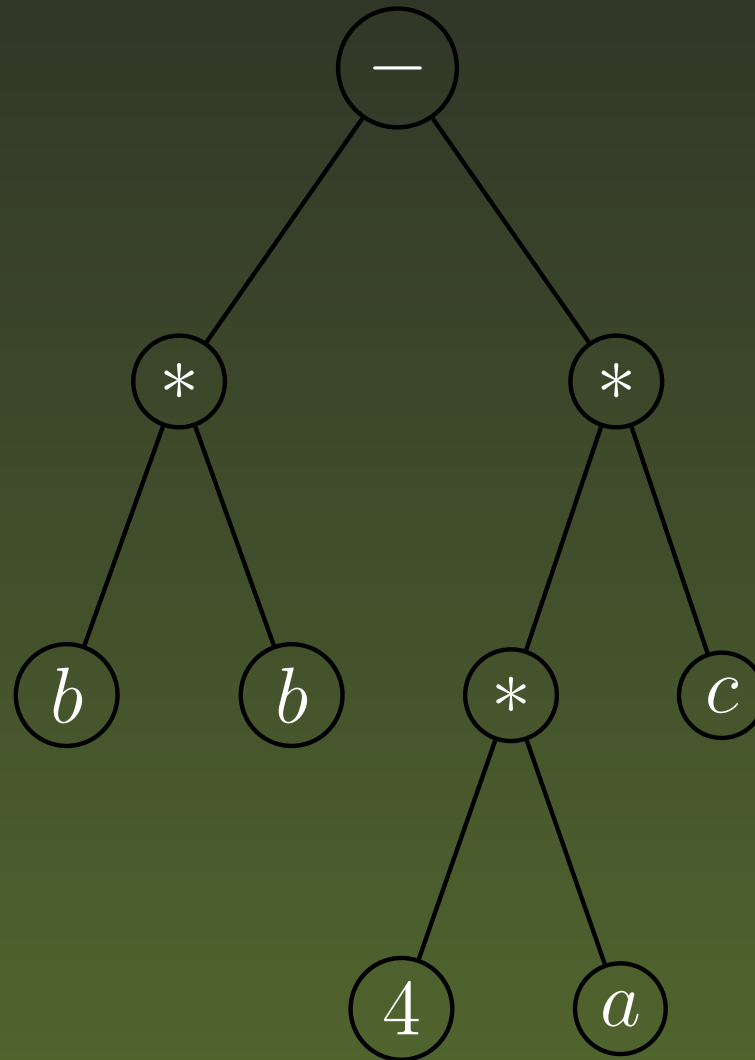
$$O \rightarrow + \mid - \mid * \mid / \quad (2)$$

$$E \rightarrow V \mid N \quad (3)$$

Szerepeljen a nyelvtan további részeiben V és N nemterminális úgy, hogy tegyék lehetővé változók és számok használatát!

Írjuk fel a $- * b b * * 4 a c$ kifejezés kifejezésfáját e nyelvtan alapján.

$- * b b * * 4 a c$



Posztfix operátorok

A posztfix operátorokat mindig az argumentumaik után írjuk, azaz a posztfix operátorok használatát leíró nyelvtani szabályokban az operátor mindig az argumentumaik után áll. Ezt nevezzük fordított lengyel jelölésnek is (*inverz Polish notation*).

Az ilyen kifejezések kifejezésfája mindig egyértelműen kitékelhető, habár nem olyan egyszerűen, mint a prefix jelölés esetében.

Posztfix algoritmus

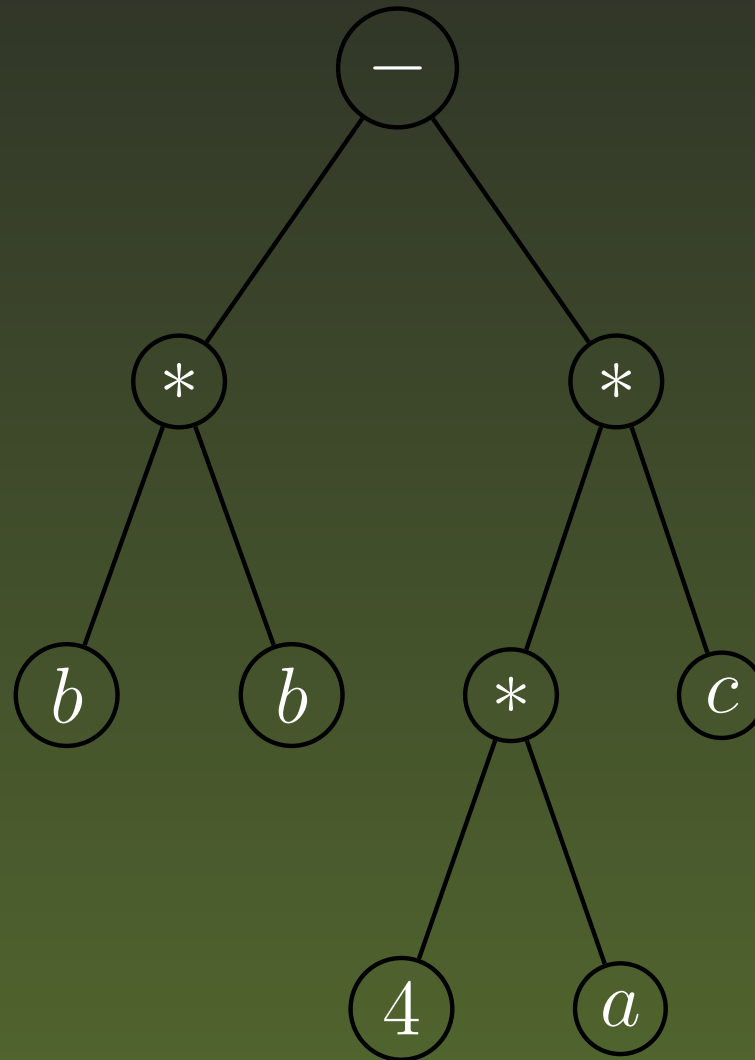
- (1) Ha nincs olvasható elem, végeztünk, ha van, legyen c az olvasott elem.
- (2) Ha c konstans vagy változó, helyezzük el a veremben.
- (3) Ha c operátor:
 - (a) Írjuk le az operátort (felülre).
 - (b) Az operandusok számának megfelelő számú veremelemet távolítsuk el és húzzuk be az éleket.
 - (c) Helyezzük el a veremben a hivatkozást a részfára, melynek csúcsa c .
- (4) Vissza az (1) pontra.

Posztfix operátorok (példa)

Legyenek egy nyelvben $-$, $+$, $*$ és $/$ posztfix operátorok!

Készítsük el a $bb * 4a * c * -$ kifejezés kifejezésfáját!
(Vegyük észre, hogy a kifejezésfa felírásához sem a prefix, sem pedig a posztfix esetben nincsen szükségünk a nyelvtanra, csak azt kell tudnunk, hogy az egyes csomópontokhoz hány él tartozik, azaz milyen az egyes elemek aritása.)

$bb*4a*c*_$



Infix operátorok

Az infix operátorok két operandusuk között állnak. (Az olyan nyelvtani szabályok, amelyekben infix operátorok használatát írják le az operátort két operandusuk közé helyezik.)

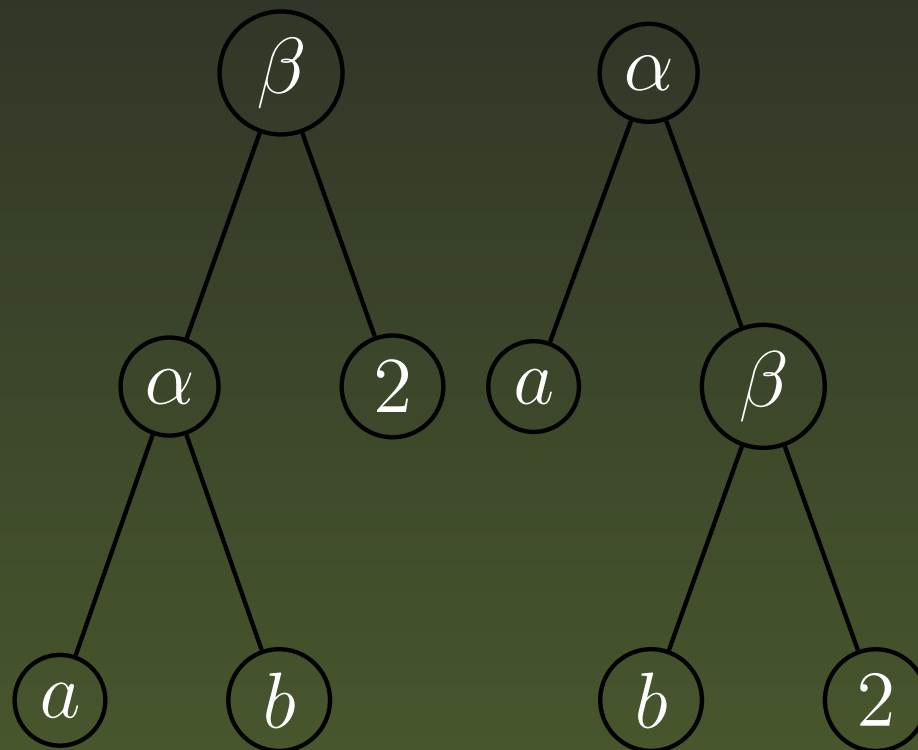
Az infix operátorokat (is) használó kifejezések kifejezésfája csak az operátorok használatára vonatkozó kiegészítő információk (precedencia) segítségével írható fel. (Kreálható volna egyértelmű algoritmus, de nem ezt tettük.)

Infix operátorok (példa)

Írjuk fel az $a \alpha b \beta$ 2 kifejezés kifejezésfáját, ahol α és β infix operátorok!

Mivel a kifejezésfa nem írható fel egyértelműen, minden lehetséges megoldást adjunk meg.

$a \alpha b \beta 2$



Precedencia

Az infix operátorok használatának egyértelművé tételéhez vezessük be a precedenciaosztályok fogalmát!

A precedenciaosztály az operátorok csoportosításának eszköze, amely a megfelelő használat mellett lehetővé teszi a kifejezésfa egyértelmű felírását.

Precedencia (példa)

Legyen egy G_p nyelvtan részlete a következő:

$$P_2 \rightarrow P_1 \mid P_1 + P_2 \mid P_1 - P_2 \quad (4)$$

$$P_1 \rightarrow E \mid E * P_1 \mid E / P_1 \quad (5)$$

$$E \rightarrow V \mid N \quad (6)$$

Legyenek a nyelvtan további szabályai úgy létrehozva, hogy V helyén álhasson változó, N helyén állandó és legyen P_2 a mondatszimbólum (startszimbólum).

Mutassuk be példákon keresztül, hogy a nyelvtan egyértelművé teszi a műveletek kiértékelésének sorrendjét az érvényes levezetéseken keresztül!

Precedencia (példa)

Mutassuk meg a $V_1 * V_2 + V_3$ kifejezés helyes levezetését G_p szerint P_2 -ből!

$$\begin{array}{c} P_2 \vdash P_1 + P_2 \vdash E * P_1 + P_2 \vdash \\ \vdash E * P_1 + P_1 \vdash E * E + E \vdash V_1 * V_2 + V_3 \end{array}$$

Láthatjuk, hogy, ha a P_2 operátorosztályba tartozó operátort előbb vezetjük be mint a P_1 operátorosztályt, a levezetés sikeres.

Precedencia (példa)

Kísérjük meg levezetni a $V_1 * V_2 + V_3$ kifejezést a P_2 nemterminálisból úgy, hogy előbb a $*$ operátort helyettesítjük be!

$$P_2 \vdash P_1 \vdash V_1 * P_1 \vdash$$

Látható, hogy a levezetés elakadt, mert a P_1 helyére már semmiképpen nem tudunk $+$ operátort behelyettesíteni.

Precedencia (példa)

Mutassuk be az $E_1 + E_2 + E_3$ kifejezés helyes levezetését G_p szerint P_2 nemterminálisból!

$$P_2 \vdash P_1 + P_2 \vdash P_1 + P_1 + P_2 \vdash \quad (7)$$

$$\vdash P_1 + P_1 + P_1 \vdash E_1 + E_2 + E_3 \quad (8)$$

Láthatjuk, hogy a $+$ operátor jobb oldalára kerülhet újabb $+$ operátor, de a bal oldalára már nem, és ez meghatározza az azonos precedenciaosztályba tartozó operátorok bevezetésének sorrendjét.

A kifejezésfa

A helyes levezetés ismeretében a kifejezésfa könnyedén felírható. A levezetést balról jobbra olvasva a bevezetett elemeket le kell írunk annak a csomópontnak a gyermekeként, amelynek helyére behelyettesítettük őket.

Egyszerű behelyettesítés esetén (pl. $P_1 \vdash E_1$) nevezzük át a megfelelő csomópontot!

Kifejezésfa (példa)

$$P_2 \vdash P_1 + P_2 \vdash E * P_1 + P_2 \vdash V_1 * V_2 + V_3$$

