

Programozás módszertan

Típuskonstrukciók

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

Bevezetés

Bevezetés

A programozási nyelvek beépített típusain kívül általában a következő típuskonstrukció-osztályok is rendelkezésünkre állnak:

- direktszorzat
- unió
- iteráló

E szakaszban a típuskonstrukciós eszközöket vesszük sorra.

Típusekvivalencia

A programozó által létrehozott típusok kapcsán felmerül a kérdés, hogy mikor azonos két típus. Ez igen fontos, hiszen a típusos nyelvekben a típusekvivalencia határozza meg, hogy mire és hol használhatjuk az adott típusú objektumokat.

A különféle nyelvek más és más módon kezelik a típusekvivalencia kérdését.

Ada

A következő programrészletben három típust hozunk létre, amelyeknek nem adunk nevet, de objektumokat létrehozunk a segítségükkel:

Forrás: Programozási nyelvek

```
1 R: array(1..10) of Integer;  
2 S, T: array(1..10) of Integer;
```

Az Ada nyelvben R, S, és T változók típusa különböző.

C

A következő C nyelvű program az előzőhöz hasonló helyzetet teremt:

```
1 struct { int x; int y; } a;  
2 struct { int x; int y; } b, c;
```

Itt az `a` típusa különböző, a `b` és `c` viszont megegyező típusúak.

Modula-3

A Modula-3 struktúrális típusequivivalenciát használ, amely két típust akkor tekint azonosnak, ha belső szerkezetük azonos.

A struktúrális típusequivivalencia csak látszólag logikus, hiszen két típust akkor is ekvivalensnek tekint, ha a valós világ teljesen különböző objektumainak leírására vonatkoznak. (Példa: állomány neve és mérete versus lépegető exkavátor tulajdonosa és tömege.)

Mutable/immutable

A mutable típusokkal létrehozott objektumok megváltoztathatók, az immutable típus garantálja a változtathatatlanságot (nem a változó konstans, a változó változtathatatlan mert a típus ezt garantálja).

Az immutable típusokkal is készíthetünk műveleteket, ezek azonban az új értéket új objektumban tárolják.

Java

Forrás: Programozási nyelvek

```
1 public final class Complex {
2     private final double re;
3     private final double im;
4
5     Complex(double re, double im){
6         this.re = re;
7         this.im = im;
8     }
9
10    public Complex add(Complex z){
11        return new Complex(re + z.re, im + z.im);
12    }
13    ...
14 }
```

Immutable

Természetesen csak ritkán, indokolt esetben használunk immutable típust, amely garantálja, hogy a létrehozott objektumok nem változnak meg.

Párhuzamos programozási környezetben immutable közös változókat használhatunk, amely garantálja, hogy az egyes végrehajtási szálak közt nem alakul ki versenyhelyzet (*race condition*).

Immutable

Az immutable típuskonstrukciós eszközöket támogató nyelvek általában nem a változókat, hanem a referenciáikat kezelik. A következő példa megváltoztatja a referenciáját:

$$a = a + 1$$

Az ilyen nyelvek általában rendelkeznek garbage collectorral, amely a példában az a változó eredeti értékének helyét – ha azt más végrehajtási szál már nem használja – felszabadítja.

Direktszorzat típus

Direktszorzat típus

A direktszorzat típus létrehozása a komponensek típusának és szelektorának felsorolásából áll.

- A felsorolt típusok már létező típusok, egy adott komponenstípus egynél többször is szerepelhet. A felsorolt típusok általában lehetnek típuskonstrukciók – pl. direktszorzat típusok – is.
- A szelektorok az adott komponens kiválasztására használhatók, minden szelektor csak egyszer szerepelhet (máskülönben nem volna egyértelmű a szelekció).

Példa

A következő példa bemutatja, hogyan hozhatunk létre direktszorzat típust C programozási nyelven.

```
typedef struct pont {  
    int x;  
    int y;  
    color szin;  
} pont;
```

Formális definíció

Legyenek $\mathcal{T}_1, \mathcal{T}_2, \dots, \mathcal{T}_n$ típusok, amelyek típusértékhalmaza T_1, T_2, \dots, T_n .

A \mathcal{T} típus a $\mathcal{T}_1, \mathcal{T}_2, \dots, \mathcal{T}_n$ típusokból képzett direktszorzat típus, ha típusértékhalmaza

$$T_1 \times T_2 \times \dots \times T_n$$

Műveletek

Szelekció

A szelektorfüggvény segítségével a direktszorzat típus komponensei választhatók ki. A kiválasztott komponens a legtöbb nyelvben belérték, azaz a szelektor segítségével a komponens megváltoztatható (a szelektor értékadás bal oldalára írható).

A direktszorzat típus használatához a szelekcióra mindenképpen szükség van, ezért azt minden programozási nyelv (amelyben van direktszorzat típus) támogatja.

Szelekció (példa)

A következő példa bemutatja, hogy a C programozási nyelvben hogyan használhatjuk a szelektor eredményét (a kiválasztott komponens) ballértékként:

```
pont a;
```

```
a.x = 40;
```

```
a.y = 2;
```

Figyeljük meg, hogy a kiválasztott komponens típusának megfelelő balértékként szerepel a kifejezésben.

Értékadás

Bizonyos nyelvek megengedik, bizonyos nyelvek pedig nem engedik meg, hogy direktszorzat típus balértékként szerepeljen az értékadásban.

(Természetesen, ha a direktszorzat típus balértékként szerepel, az adott értékadásban a jobb oldalon is olyan objektum állhat, amely direktszorzat típusú vagy direktszorzat típusra alakítható).

C

A C programozási nyelvben a direktszorzat típus balértékként szerepelhet, a használatakor a fordító által készített kód a direktszorzat típusú objektum szolgálai lemásolását tartalmazza.

A fordító minden esetben ismeri a direktszorzat típusú objektum méretét, így képes olyan kódot előállítani, amely az objektumot alkotó memóriaterület lemásolását végzi.

A másolás a C nyelvben a belső szerkezet figyelembevétele nélkül történik.

C (példa)

A következő példa bemutatja, hogyan használhatjuk a direktszorzat típusú objektumokat értékadásban.

```
pont a, b;  
a.x = 40;  
a.y = 2;  
  
b = a;
```

Ada

Az Ada nyelvben a programozó megtilthatja, hogy az adott direktszorzat típusához tartozó objektum értékadásban szerepeljen.

Ha egy direktszorzat típus valamely elemére az értékadás tiltott, az tiltott lesz az adott típusra is.

Az értékadás a C nyelvben nem művelet, hanem operátor és ez így van a C++ nyelvben is.

A C++ nyelv lehetővé teszi az operátorok túlterhelését (*operator overloading*), így a programozó az adott típushoz elkészítheti az értékadást végző programot, azaz meghatározhatja, hogy mi történjék az értékadó operátor hatására.

Konstansok

Bizonyos nyelvek megengedik, hogy direktszorzat típusú literálokat használjunk. Az ilyen literálok neve *aggregátum*.

Bizonyos nyelvekben az aggregátumok használhatók a direktszorzat típusú objektumok kezdeti értékének beállítására (inicializálásra).

Konstansok (példa)

A következő példa bemutatja hogyan használhatók aggregátumok változóinicializálásra a C programozási nyelvben:

```
pont a = { 40, 42, red } ;
```

A C programozási nyelvekben tehát az aggregátumok { } jelek közt felsorolt elemek, amelyek sorrendje alapján rendelődnek az egyes komponensekhez.

Egyenlőségvizsgálat

A direktszorzat típus esetén nem végezhető el az egyenlőségvizsgálat a memóriaterületek egyszerű összehasonlításával, ezért sok programozási nyelv nem támogatja az ilyen objektumok összehasonlítását.

A nehézséget az okozza, hogy a fordító a típus komponensei közé kitöltőbájtokat helyez (*padding*), hogy az egyes komponensek szóhatárra essenek és ezeket a kitöltőbájtokat nem inicializálja.

(Az is következik a padding használatából, hogy a direktszorzat típusú objektumok mérete nem egyezik meg a komponensek méretének összegével!)

Egyenlőségvizsgálat (példa)

A C programozási nyelvben direktszorzat típusú objektumokat nem hasonlíthatunk össze, esetükben az egyenlőségvizsgálatot komponensenként kell elvégeznünk.

```
if ( a.x == b.x &&  
    a.y == b.y &&  
    a.color == b.color )  
    ...
```

Típus inicializálás

Egyes nyelvek (pl. Ada) lehetővé teszik, hogy a direktszorzat típus deklarációjakor előírjuk a mezők kezdeti értékét. Később, amikor az adott típussal objektumot hozunk létre, az adott mezők a megfelelő értékre lesznek állítva.

Ezzel az eszközzel típusinvariánst fogalmazhatunk meg az általunk létrehozott típusra, ami igen hasznos.

Az unió típus

Az unió típus

Az unió típus esetén kettő vagy több típus azonos memóriaterületen való tárolását tesszük lehetővé (egyszerre mindig csak egyféle értéket tárolhatunk).

Amíg a direktzorzat típussal a komponenseknél összetettebb adatszerkezetet hozunk létre, addig az unió típussal a komponenseknél általánosabb adatszerkezetet készítünk.

Formális definíció

Legyenek $\mathcal{T}_1, \mathcal{T}_2, \dots, \mathcal{T}_n$ típusok és legyen típusérték-halmazuk T_1, T_2, \dots, T_n .

A \mathcal{T} típus a $\mathcal{T}_1, \mathcal{T}_2, \dots, \mathcal{T}_n$ típusokból képzett unió típuskonstrukció, ha típusérték-halmaza

$$T = T_1 \cup T_2 \cup \dots \cup T_n$$

Unió típus (példa)

A következő példa bemutatja, hogyan készíthetünk olyan típust a C programozási nyelvben, amely egész és lebegőpontos szám ábrázolására is alkalmas:

```
typedef union szam {  
    double d;  
    int i;  
} szam;
```


Unió típus

A régebbi nyelvek nem támogatják az unió típust, míg a modern nyelvek modernebb eszközöket ajánlanak az unió típus helyett.

Általánosságban elmondható, hogy az unió típuskonstrukciót sokkal ritkábban használjuk mint a direktszorzat típuskonstrukciót.

Műveletek

Az unió típusok kezelése, a velük kapcsolatban használható műveletek igen változatosan alakul a különféle nyelvekben.

Az egyes nyelvek különféle módon kezelik az unió jellegű típuskonstrukciókat, különféle unió típusokat figyelhetünk meg.

Kötött/szabad

Ha egy unió típusú objektumban elhelyezzük az egyik komponensnek megfelelő értéket, az objektumban tárolt adat típusa megváltozik. Az unió típusú objektum típusa tehát tulajdonképpen csak futáskor ellenőrizhető.

Ezt a problémát a nyelvek kétféleképpen kezelhetik:

- Futási idejű típusellenőrzéssel (kötött unió).
- Az ellenőrzés elhagyásával (szabad unió).

Algol 68

Forrás: Programozási nyelvek

```
1 mode union (int, real) NumberType;  
2 NumberType number;  
3 ...  
4 number := 3.14;  
5 ...  
6 case number in  
7   (int i): print(("integer", i)),  
8   (real r): print(("real", r))  
9 esac
```

Figyeljük meg, hogy a típusellenőrzés futási időben elvégezhető!

C

A C programozási nyelvben az union szelektora nem annak eldöntésére szolgál, hogy az adott területre milyen típusú adatot helyeztünk, hanem annak jelzésére, hogy milyen típusú adatként kívánjuk kiolvasni az ott elhelyezett adatot!

```
1  typedef union {  
2      int i;  
3      double d;  
4  }  szam;  
5  
6  szam a;  
7  
8  a.i = 10;  
9  a.d = a.d + 1;
```

C

A C union típusokban elhelyezett adatok kezeléséről a programozónak kell gondoskodnia, amelyet a következő trükkel lehet kényelmesen megoldani.

Forrás: Programozási nyelvek

```
1 typedef struct {  
2     NemTipus nem;  
3     union {  
4         FerfiTipus ferfi;  
5         NoTipus no;  
6     } adat;  
7 } EmberTipus;
```

A programozóra hárul a feladat, hogy a nem komponens értékében folyamatosan nyilvántartsa mit helyezett el az adat komponensben.

C

```
1  typedef struct {  
2      int tipus;  
3      int ertek;  
4  } egesz;  
5  
6  typedef struct {  
7      int tipus;  
8      double ertek;  
9  } valos;  
10  
11 typedef union {  
12     egesz a;  
13     valos b;  
14 } szam;
```

Iterált típus

Iterált típus

A legtöbbet használt és legérdekesebb típuskonstrukció az iterált típus. A különféle nyelvek több változatát is támogatják, melyek közül a legfontosabbak:

- vektor (egydimenziós tömb)
- kettő vagy több dimenziós tömb
- lista
- hasítótábla
- sor
- fájl
- halmaz

Iterált típus

Az iterált típuskonstrukció minden változata homogén, azonos típusú komponensekből képez újabb típust.

(A tömb elemeinek mindegyike azonos típusú, de ez lehet összetett típus is, így a tömbben nem homogén adatok is lehetnek.)

(a) Vektor

Vektor

A vektor típus létrehozásakor meg kell adnunk az elemek típusát és meg kell határoznunk az indexhalmazt. Az indexhalmaz megadása az egyes nyelvekben különböző lehet.

Basic

A Basic nyelv különlegessége, hogy $0, \dots, 10$ indexhalmazzal definíció nélkül használhatunk vektorokat. Mivel a változók létrehozása automatikus, a következő program működőképes:

```
1 10 let a(4) = 18
2 15 let a(3) = 42 - a(4)
3 20 print (a(4) + a(3))
```

C

A C programozási nyelvben és a leszármazottaiban a vektortípus létrehozásakor a vektor méretét kell megadnunk.

A vektorelemek indexelés 0-tól kezdődően pozitív egész számokkal történik, a legnagyobb indexértéke eggyel kisebb mint a definíciókor megadott méret. Az n méretű tömb indexértékei tehát $0 \leq i \leq n - 1$.

Szelekció

A szelekció segítségével a vektor típusú objektum egy elemét választjuk ki az index megadásával. A kiválasztott elem a különféle programozási nyelvekben balérték, értékadás bal oldalán szerepelhet.

A vektor v_i eleme a legtöbb programozási nyelvben $v[i]$ vagy $v(i)$ formában választható ki a szelekció műveletével.

Értékadás

Sok programozási nyelv nem teszi lehetővé, hogy vektor típusú objektumok értékadásban szerepeljenek. Az ilyen nyelvekben egyszerű ciklus segítségével a programozó végezheti el az értékek másolását elemenként.

Más nyelvek lehetővé teszik a vektorok értékadásban való használatát. Az ilyen nyelvek (Pascal) nyilvánvalóan a vektor méretét is nyilvántartják, azaz a vektor mégsem egészen homogén adatszerkezet.

C

A C programozási nyelvben (és leszármazottaiban) a vektor méretét nem tartjuk nyilván, a vektor csak az elemek tartalmazza, ezért nem is szerepelhet értékadásban.

A vektorok kezelésekor a C garantálja, hogy bájtfolysónosan a megfelelő lineáris címmel helyezi el a vektorelemeket, ezért használható a már ismerttetett mutatóaritmetika.

A C nyelvben az indexelés operátora a `[]` posztfix operátor, indexhatárelenőrzés nincs.

A C++ nyelv lehetővé teszi az operátor túlterhelést, így az indexelés megváltoztatható. Akár indexhatárelőrzést is készthetünk.

Java

A Java nyelvben a vektorok méretét nyilvántartjuk, minden vektor mérete lekérdezhető.

A méret nyilvántartása fontos, hiszen a Java vektorai dinamikusak, futás közben a vektor mérete szabadon változtatható.

Ada és Fortran

Az Ada vektorai nem dinamikusak, de alprogram létrehozásakor használhatunk ismeretlen méretű tömböt fogadó formális paramétert.

Ez a tulajdonság a Fortran nyelvben is adott, az alprogramokon belül ott is használhatunk olyan vektorokat, amelyek mérete csak futási időben válik ismertté. A Fortran nyelvben azonban minden dinamikus tömb valamely statikus tömb másolata.

(b) Tömbök

Tömbök

A tömbök a vektortípus általánosításai, ahol az indexben több indexérték is szerepelhet (pl. $t_{i,j}$).

Sok programozási nyelv nem támogatja a több dimenziós tömök készítését, hiszen azok vektorok segítségével is ábrázolhatók. A két dimenziós tömb például vektorokból készített vektorként ábrázolható.

A címfüggvény

A vektorok természetes formájukban ábrázolhatók, hiszen a vektor címezéséhez is és az operatív memória címezéséhez is egy indexet használunk. A tömb és a memória egydimenziósak.

Többdimenziós tömbök esetén a *címfüggvény* a tömb indexeit fordítja le memóriacímekre.

Egy dimenzió

Legyen T egydimenziós tömb (vektor), melynek elemszáma n , (címezése $0 \leq i \leq n - 1$ értékekkel történjen). A T_0 memóriacíme (a vektor kezdőcíme) legyen t , egy elem mérete pedig s .

Akkor a T_i elem címét megkapjuk az

$$f(i) = t + is$$

címfüggvény segítségével.

Két dimenzió (sorfolytanos)

Legyen T kétdimenziós tömb, elemszáma $n_1 \times n_2$, a $T_{0,0}$ (első elem) címe legyen t , az elemek mérete pedig s .

Akkor a $T_{i,j}$ elem címét megkapjuk az

$$f(i, j) = t + s(i + jn_1)$$

címfüggvény segítségével.

Három dimenzió

Legyen T három dimenziós tömb, elemszáma $n_1 \times n_2 \times n_3$, a $T(0, 0, 0)$ (első elem) címe legyen t , az elemek mérete pedig s .

Akkor a $T_{i,j}$ elem címét megkapjuk az

$$f(i, j, k) = t + s(i + jn_1 + kn_1n_2)$$

címfüggvény segítségével.

Sorfolytonos címfüggvény

Legyen T egy k dimenziós n_1, n_2, \dots, n_k méretű tömb,
legyen $p_1 = 1$ és $2 \leq j \leq k$ értékekre

$$p_j = \prod_{i=1}^{j-1} n_i$$

Ha a tömb kezdőcíme t és egy elem mérete s , megkapjuk
a T_{i_1, i_2, \dots, i_k} elem címét az

$$f(i_1, i_2, \dots, i_k) = t + s \left(\sum_{j=1}^k i_j p_j \right)$$

címfüggvénnyel.

C

A C programozási nyelv nem támogatja több dimenziós tömbök készítését, használhatunk viszont vektorokból épített vektorokat. Ezt mutatja be a következő programrészlet:

```
1  int t[100][10];  
2  int i, j;  
3  
4  for(i = 0; i < 100; ++i)  
5      for(j = 0; j < 10; ++j)  
6          t[i][j] = 42;
```

FORTRAN

A FORTRAN szabvány a kezdeti időkben minimum három, később minimum 7 dimenziós tömbök kezelését tették kötelező a fordítók számára.

```
1      INTEGER T(100,10)
2      DO 10 I = 1, 100
3      DO 10 J = 1, 10
4      T(I,J) = 42
5      10 CONTINUE
```

További konstrukciók

Iterált típus

Ejtsünk néhány szót a további iterált típuskonstrukciókról. A teljes lista:

- vektor (egydimenziós tömb)
- kettő vagy több dimenziós tömb
- lista
- hasítótábla
- sor
- fájl
- halmaz