

**Primitív adattípusok és implementációjuk (egész, lebegőpontos, komplex, decimális, logikai, karakter); karakterláncok megvalósítási lehetőségei, műveletei különböző nyelvekben; felhasználó által definiált felsorolt típusok; tartomány és tömb típusok**

**Adattípus:** Az adatabsztrakció első megjelenési formája a programozási nyelvekben. Az adattípus maga egy absztrakt programozási eszköz, amely mindig más, konkrét programozási eszköz egy komponenseként jelenik meg. Az adattípusnak neve van, ami egy azonosító. A programozási nyelvek egy része ismeri ezt az eszközt, más része nem. Ennek megfelelően beszélünk típusos és nem típusos nyelvekről. Az eljárás-orientált nyelvek típusosak. Egy adattípust három dolog határoz meg, ezek:

–*tartomány:* azokat az elemeket tartalmazza, amelyeket az adott típusú konkrét programozási eszköz fölvehet értékként.

–*műveletek* (amelyeket a tartomány elemein végre tudunk hajtani)

–*reprezentáció:* egy belső ábrázolási mód. Az egyes típusok tartományába tartozó értékek tárban való megjelenését határozza meg, tehát azt, hogy az egyes elemek hány bájtira és milyen bitkombinációra képződnek le.

Minden típusos nyelv rendelkezik beépített (standard) típusokkal. Egyes nyelvek lehetővé teszik azt, hogy a programozó is definiálhasson típusokat. Ez az adatabsztrakciónak egy magasabb szintjét jelenti, segítségével a valós világ egyedeinek tulajdonságait jobban tudjuk modellezni.

**Primitív (=skalár/egyszerű) adattípus:** tartománya atomi értékeket tartalmaz, minden érték egyedi, közvetlenül nyelvi eszközökkel tovább nem bontható.

Az egész és valós típusokra közös néven, mint numerikus típusokra hivatkozunk. A numerikus típusok értékein a numerikus és hasonlító műveletek hajthatók végre.

*Egész típus:* Egy vagy több típusa minden nyelvben létezik. Ezek belső ábrázolása fixpontos. Az egyes egész típusok az ábrázoláshoz szükséges bájtok számában térnek el és nyilván ez határozza meg a tartományukat is. Néhány nyelv ismeri az előjel nélküli egész típust, ennek belső ábrázolása előjel nélküli (direkt) történik.

*Valós típusok:* Belső ábrázolásuk lebegőpontos. A tartomány itt is az alkalmazott ábrázolás függvénye, ez viszont általában implementációfüggő. *Karakteres típus:* tartományának elemei karakterek, a karakterlánc vagy sztring típusú pedig karaktersorozatok. Ábrázolásuk karakteres (karakterenként egy vagy két bájt, az alkalmazott kódtáblától függően), műveleteik a szöveges és hasonlító műveletek.

*Logikai típus:* tartománya a hamis és igaz értékekből áll, műveletei a logikai és hasonlító műveletek, belső ábrázolása logikai.

*Felsorolásos típus:* Speciális egyszerű típus. A felsorolásos típust saját típusként kell létrehozni. A típus definiálása úgy történik, hogy megadjuk a tartomány elemeit. Ezek azonosítók lehetnek. Az elemekre alkalmazhatók a hasonlító műveletek.

**String adattípus:** olyan adattípus, amely egy ideális formális stringet modellez. Az egyik legfontosabb és leggyakrabban használt adattípusok, ezért lényegében minden programozási nyelvben létezik valamilyen megvalósításuk. Néhány nyelvben az elemi típusok közé tartozik, más nyelvek az

összetett típusok közé sorolják. A legtöbb magas-szintű programozási nyelv szintaxisa megengedi a string használatát, általában valamilyen idézőjeles formában, és a string adattípus egy bizonyos megjelenéseként tekintik; a meta-stringre viszont gyakran literál vagy string literál kifejezésekkel hivatkoznak.

Annak ellenére, hogy a formális string tetszés szerinti (de véges) hosszúságú, a megvalósított nyelvekben ez a hossz gyakran mesterségesen maximalizált. Általánosságban, két fajtája létezik a string adattípusnak: *fixhosszúságú stringek*, amikor a függetlenül attól, hogy a string elemeinek száma eléri-e a maximumot, mindig ugyanakkora memóriaterületet foglalnak le a tárolásához, valamint *változó hosszúságú stringek*, ahol a string hossza nem fix, és csak az aktuális hossza szerinti helyet foglal el memóriában. (modern programozási nyelvekben a legtöbb string változó hosszúságú)

Történetileg, a string adattípus esetében egy byte tartozott egy-egy karakterhez, annak ellenére, hogy a különböző karakter kódtáblák régióként változtak, a tényleges tartalom gyakorlatilag a programozóktól függött. Később több kódtáblát használtak azonos régiók, így a programozók egyre gyakrabban szegték meg az egy karakter-egy byte szabályt.

A logografikus nyelvek esetében (mint például a kínai, a japán és a koreai (=“CJK”)) szükségessé vált a 256-nál nagyobb, azaz a több mint 1 byte-os kódok használata. A megoldást az jelentette, hogy a megvalósítás során 1 byte-os kódokat használtak az ASCII karakterek esetén, és két byte-os kódokat a CJK szóképek esetén. Ez a megoldás azonban több szempontból is problematikus: nem lehet a stringeket összehasonlítani és darabolni, csak ha pontosan ismert, milyen a tárolás kódja, de ebben az esetben is kérdéses az összehasonlítás a különböző kódolású stringek között, valamint a rendezés.

A Unicode csak tovább bonyolítja a helyzetet. A legtöbb nyelvben létezik adattípus Unicode stringekre (általában az UTF-16, amit a Unicode előtt használtak). A Unicode és a lokális kódolás közötti konverzióhoz alapvetően szükséges a helyi kódolás alapos ismerete, de így is gondot okozhat, ha különféle módon kódolt stringeket adatátviteli vonalon keresztül továbbítanak, olyan jelzés nélkül, hogy milyen kódolású az adott string.

*Megvalósítása:* erősen függ attól, hogy milyen karakter készletet használ és milyen kódolást támogat a megvalósítás környezete. A legtöbb nagyon hasonlít egy változó hosszúságú tömb megvalósításához, ahol az elemek tárolása karakter kódok segítségével történik. A fő különbség az, hogy a string esetében logikailag csak bizonyos kódolású karakterek tárolása megengedett. Tulajdonképpen ez történik, ha UTF-8 kódolást használnak, ekkor egy karakter egy és négy byte közötti helyet foglal el. Ebben az esetben például a string hossza különbözik a tömb hosszától.

A string hossza tárolható úgy, hogy implícit módon egy speciális záró karaktert használnak; ez a karakter általában a null karakter, amelynek értéke nulla, ezt a konvenciót használja a C. Ekkor a string hossza számszerűen nincsen tárolva, a string végét egy egyértelműen felismerhető elválasztó/lezáró karakter jelzi. Más megvalósítás esetén a string hosszát is számszerűen tárolják, általában byte-okban megadott hossza megelőzi a string karaktereit – ezt a konvenciót használja a Pascal.

A karakterrel lezárt stringek esetében szabály, hogy a lezáró karakter nem lehet olyan karakter, amely előfordulhat a stringben.

## **Tömb típus:**

Az eljárásorientált nyelvek két legfontosabb összetett típusa (melyet minden nyelv ismer) a **tömb** és a **rekord**. A tömb típus a tömb absztrakt adatszerkezet megjelenése típus szinten. A tömb statikus és homogén összetett típus, vagyis tartományának elemei olyan értékcsoporthoz tartoznak, amelyekben az elemek száma azonos, és az elemek azonos típusúak. A tömböt, mint típust meghatározza a dimenzióinak száma, az indexkészletének típusa és tartománya, valamint az elemeinek a típusa.

Egyes nyelvek (pl. a C) nem ismerik a többdimenziós tömböket, és ezeket úgy képzelik el, mint olyan egydimenziós tömbök, amelyek elemei egydimenziós tömbök. Reprezentációjuk lehet sor- vagy oszlopfolytonos. Ez általában implementációfüggő, a sorfolytonos a gyakoribb.

Ha van egy tömb típusú programozási eszközünk, akkor a nevével az összes elemre együtt, mint egy értékcsoporthoz tudunk hivatkozni (az elemek sorrendjét a reprezentáció határozza meg). Az értékcsoporthoz egyes elemekre a programozási eszköz neve után megadott indexek segítségével hivatkozunk. Az indexek a nyelvek egy részében szögletes, másik részében kerek zárójelek között állnak. Egyes nyelvek (pl. COBOL, PL/I) megengedik azt is, hogy a tömb egy adott dimenziójának összes elemét (pl. egy kétdimenziós tömb egy sorát) együtt hivatkozhatjuk.

nyelveknek a tömb típussal kapcsolatban a következő kérdéseket kell megválaszolniuk:

### *Elemtípusok:*

- Minden nyelv bármelyik skalár típust megengedi.
- A modernebb nyelvek összetett típusokat is megengednek.

### *Index lehetséges típusa:*

- Minden nyelv megengedi, hogy egész típusú legyen.
- A Pascalban és az Adában sorszámozott típusú is lehet.

### *Indextartomány megadása tömb típus deklarációsakor:*

- Lehet intervallum típusú értékkel (pl. Pascal, Ada), azaz meg kell adni az alsó és a felsőhatárt.
- Más nyelveknél (pl. PL/I) az indextartomány alsó határa a nyelv által rögzített (általában 1), és csak a tartomány felső határát kell megadni.
- A nyelvek egy szűkebb csoportja szerint csak a felső határt kell megadni, de az alsót nem a nyelv rögzíti, hanem a programozó.
- Ritkán (pl. C) az adott dimenzióban lévő elemek darabszámát kell megadni, az indexek tartományát ez alapján a nyelv határozza meg.

### *Alsó és a felső határt, illetve a darabszám megadása:*

- Literállal vagy nevesített konstanssal (pl. FORTRAN, COBOL, Pascal), vagy konstans kifejezéssel (pl. C). Ezek a statikus tömbhatárokkal dolgozó nyelvek. Itt fordítási időben eldől az értékcsoporthoz tartozó elemek darabszáma.
- Kifejezéssel. (pl. PL/I, Ada). Ezek a dinamikus tömbhatárt alkalmazó nyelvek. Itt futási időben dől el a darabszám, de a rögzítés után az természetesen nem változik.