



Programozási nyelvek és módszerek

2. ELŐADÁS – LEXIKAI ELEMEEK, BEÉPÍTETT
TÍPUSOK

Programozási nyelv megadása

Programozási nyelv megadása

Formális szintaxis leírás – ezt használja a fordítóprogram

Formális szemantika leírás – a jelentés
ellentmondásmentességének ellenőrzésére

John Backus (1924-2007)



FORTRAN tervezője

"Munkám túlnyomó része egyszerűen a lustaságomból eredt. Amikor a rakéták röppályáját kiszámító programokon dolgoztam, hozzáfogtam egy programozási rendszer kidolgozásához, amelynek elsődleges feladata a munka megkönnyítése, leegyszerűsítése volt"

Backus-Naur Form (BNF)

Nyelv a programozási nyelvek definiálására

- meta-nyelv, eredetileg 59-60-ból, az Algol és Fortran definiálására
- változatok ma is programozási nyelvek megadására
- <http://cui.unige.ch/db-research/Enseignement/analyseinfo/BNFweb.html>

BNF

ha egy szintaxis BNF formában adott, vannak eszközök, melyek automatikusan generálnak egy elemzőt, mely a szintaktikus helyességet ellenőrzi, és levezetési fát produkál, sőt le is fordítja a programot

<http://www.thefreecountry.com/developercity/constructcompilers.shtml>

Nézzük meg a Java BNF nyelvleírását!

Fordítóprogram

Forrásprogram $\Rightarrow$

Forrás-kezelő (source handler)



Lexikális elemző (scanner)



Szintaktikus elemző (parser)



Szemantikus elemző (semantic analyzer)



Belső reprezentáció



Kódgenerátor (code generator)

$\Rightarrow$ Tárgyprogram

Fordítóprogram (folyt.)

3. Lexikális elemzés: megadható reguláris nyelvtannal
2. Szintaktikus elemzés: megadható környezetfüggetlen nyelvtannal
1. Szemantikus elemzés: környezetfüggő
 - pl. egy változó használatának helyessége függhet a deklarációjától, azaz a környezettől)

A három lépés szétválasztásának oka

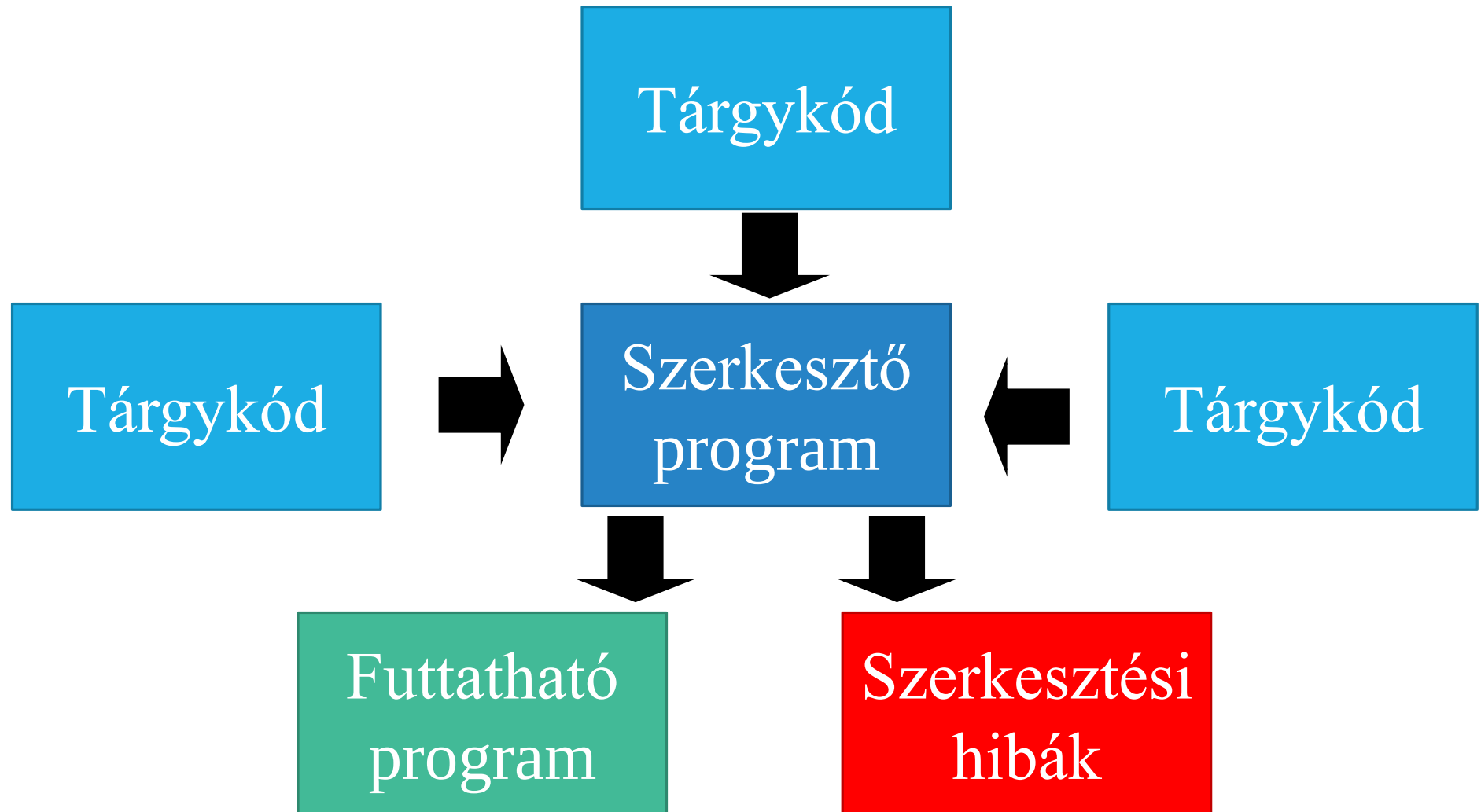
- Egyszerű feladathoz ne használjunk bonyolult eszközöket.

Programszerkezet

Fordítási egység vagy modul: A nyelvnek az az egysége, ami a fordítóprogram egyszeri lefuttatásával, a program többi részétől elkülönülten lefordítható

Több modulból álló programok esetében a fordítás és a végrehajtás között: a program **összeszerkesztése**

Szerkesztés



Programegység, fordítási egység

A **programegység** egy részfeladatot megoldó, tehát funkcionálisan összefüggő programrész.

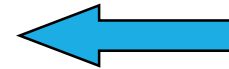
A **fordítási egység** programegységek halmaza.

A programegységek részei

A specifikációs rész írja le az egységnek a más egységből elérhető "kapcsolódási pontjait".

A törzs tartalmazza az egység funkcióját megvalósító programot.

```
procedure Hello;
```



specifikáció

```
with Text_IO; use Text_IO;
```

```
procedure Hello is
```

```
    S: String(1..40):=(others=>' ');
```

```
    Last: Natural;
```

```
begin
```

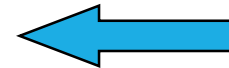
```
    Put("Hogy hívnak?");
```

```
    Get_Line(S, Last);
```

```
    New_Line;
```

```
    Put("Hello "&S(1..Last));
```

```
end Hello;
```



törzs

A törzs felépítése

A deklarációs rész (ha van külön) azonosítók **deklarációit**, valamint konstansok, típusok, objektumok és programegységek **definícióit** tartalmazhatja.

Az **utasítássorozat** tartalmazza azokat az utasításokat, amelyek **végrehajtása** nyomán a programegység kifejti a hatását.

Deklarációk

A deklarációs rész feldolgozását, a definíciók értelmezését a **deklarációs rész kiértékelésének** mondjuk.

A deklarációs rész definícióinak kiértékelése történhet:

- Fordítási (pl, C, C++) vagy
- Futási (pl. PL/I, Ada) időben.

Az első esetben **statikus definíció-kiértékelés**,

A másodikban **dinamikus definíció-kiértékelés**.

Deklarációk (folyt.)

Változódeklaráció – sokszor definíció is

- állapottér megadásához
- Formája általában
 - típus azonosító
 - `int i;`
- Vagy
 - azonosító: típus
 - `N: Natural;`

Hatáskör, láthatóság

A programszövegnek azt a szakaszát, amelyen belül az adott deklaráció érvényben van, a deklaráció **hatáskörének** nevezzük.

A programszövegnek azt a szakaszát, ahol a deklarált azonosítóval az adott deklarációra hivatkozni lehet, a deklaráció **láthatósági körének** nevezzük.

Blokkstruktúra

A **blokkstruktúra** a programegységek egymásba ágyazásával előálló hierarchikus struktúra.

Blokkstruktúrában egy programegység deklarációinak hatásköre kiterjed az összes **tartalmazott** programegységre is.

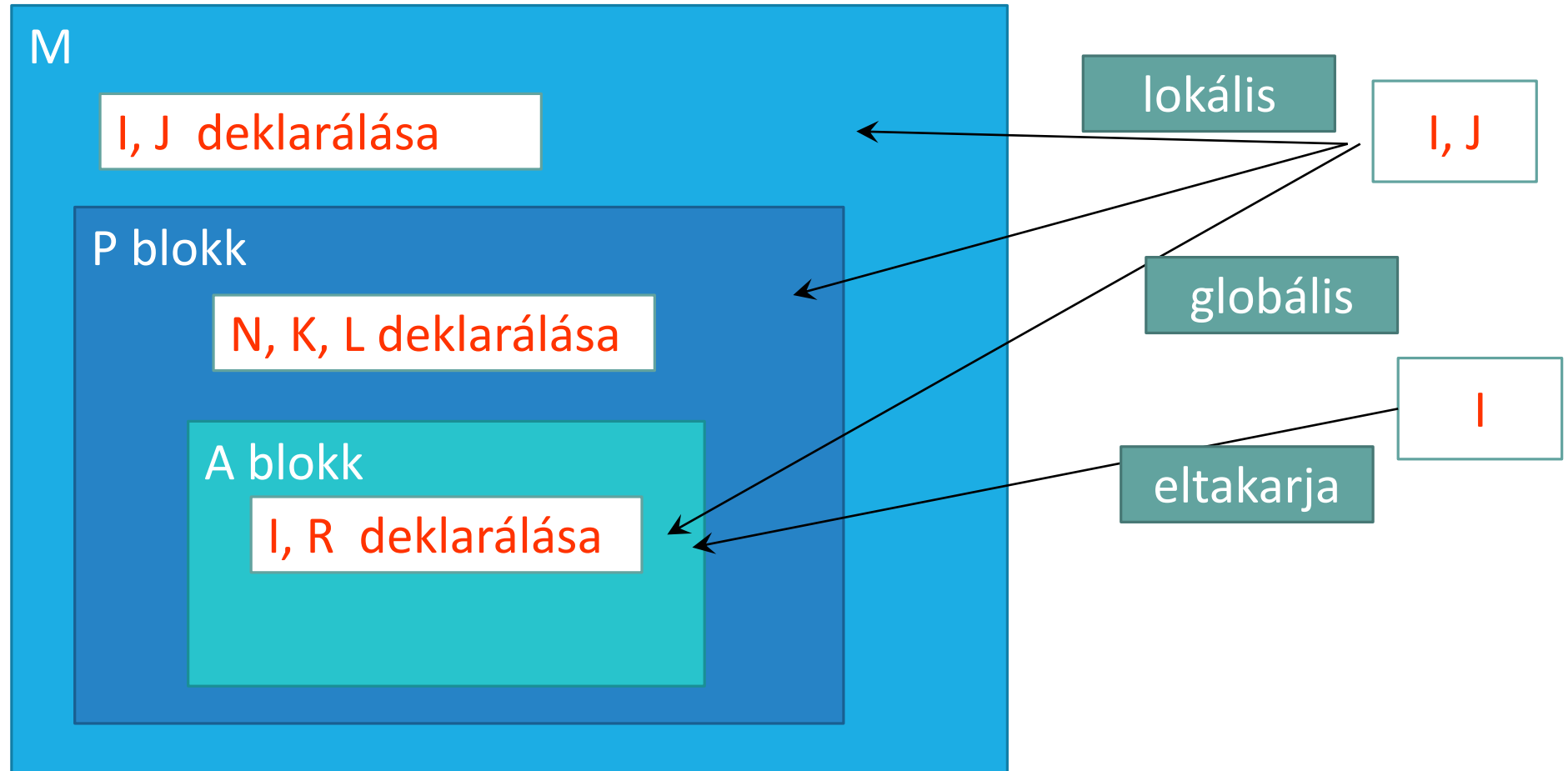
Globális – lokális

Egy programegységben a tartalmazó programegységek által deklarált azonosítókat (a tartalmazott programegység szemszögéből) **globális azonosító**knak nevezzük.

A programegységben deklarált azonosítókat magában a programegységben **lokális azonosító**knak nevezzük.

Blokkstruktúra esetén egy programegységben egy globális azonosító újradeklarálható, ekkor a lokális deklaráció **eltakarja** a globális deklarációt.

Példa



Példa

```
procedure M is
```

```
  I, J:Integer;
  procedure P is
    N, K, L:Integer;
    procedure A is
      I, R:Integer;
      begin
```

```
        I:=3; J:=9;
        N:=1;
        M.I:=4;
```

```
      end;
```

```
    begin
      I:=5;
      A;
```

```
  end;
```

```
begin
  I:=3;
  P;
```

```
end;
```

```
-- A-beli I, M-beli J
-- P-beli N
-- minősített hivatkozás!
-- ...
```

```
-- M-beli I
-- hívja A-t
-- ....
```

```
-- hívja P-t ...
```

Azonosító túlterhelése

Egy azonosítónak ugyanabban a szövegrészben több definíciója is érvényben lehet

A túlterhelést a leggyakrabban alprogramok nevei esetében teszik lehetővé – általában a paraméterek segítségével tudjuk megkülönböztetni

- `write(2)`
- `write("ez valami más")`
- `2 + 3`
- `2.1 + 3.0`

Erősen típusos nyelv

Egy nyelv **erősen típusos**, ha minden érték, objektum, formális paraméter és függvény típusa **fordítási időben egyértelműen meghatározható**.

Statikus típusosság, ha a típusok ellenőrzése fordítási időben történik, míg a **dinamikus típusosság** esetén a típusellenőrzések (műveletekhez) futás időben történnek.

Az erősen típusos nyelv és a statikus típusosság összefonódása ebből következik.

Memóriakezelés

A változóhoz a memóriaterület hozzárendelése (allokálása) történhet:

- automatikusan, a definíció kiértékelésekor, vagy
- a programozó rendelkezik róla

a változó által lefoglalt terület felszabadítása lehet:

- automatikus, vagy
- a programozó hatáskörébe tartozik.

Élettartam

A változókhoz a szükséges memóriaterület lefoglalása (allokálása) és annak felszabadítása közötti időt a változó **élettartam**ának nevezzük.

Statikus változó

élettartama a program egész működése idejére kiterjed
elhelyezése mindig az ún. statikus (main) memória-
területre történik

a statikus terület egyszer, a program betöltésekor kerül
lefoglalásra

Dinamikus változó

a program explicit módon foglal le területet, a **dinamikus** (heap) **memória**területen, amire a címével, úgynevezett **pointer**rel lehet hivatkozni.

egyes nyelvek tartalmazznak utasítást a dinamikus területek felszabadítására is.

blokkstruktúrában a programegységek (nem statikus) lokális változói az úgynevezett **végrehajtási verem**ben helyezkednek el.

Paraméterátadás

A programegységek legelterjedtebb fajtája az alprogram. A nyelvek kétféle alprogramot használnak, ezek az eljárások és a függvények.

Az alprogramoknál a hívóval való kommunikációt tervezéskor formális paraméterekkel írjuk le, amelyeknek az alprogram aktivizálásakor (hívásakor) aktuális paramétereket kell megfeleltetni.

Később ...

Lexikális elemek

A fordítási egység

A programok fordítási egységei lexikális elemek sorozatai.

A lexikális elemek karaktersorozatok, határoló jelekkel elválasztva.

A nyelvekben általában megkülönböztetjük

- F grafikus karaktereket
 - a Latin-1 vagy az angol abc betűi, számjegyek és speciális karakterek
 - Például: " # & ' () , * - + / : ; < = _ [] { } | . -
- Formátum vezérlő karaktereket
 - Például az ISO 6429 szabványban:
 - character tabulation (HT)
 - line tabulation (VT)
 - carriage return (CR)
 - line feed (LF)
 - and form feed (FF)
- Az egyéb vezérlő karaktereket
 - Ez implementáció függő

Lexikális elemek

Lexikális elem:

- Azonosító
- Numerikus literál
- Karakter literál
- String literál
- Határoló
- Megjegyzés

A jelkészlet

A karakterkészlet különböző karakterek egy halmaza.

- Nincs feltételezésünk a belső ábrázolásáról, rendezettségéről.
- Megadjuk a karakterek neveit és a karakter megjelenítését látható formában.
- Tartalmazhat olyan karaktereket, amelyek bizonyos megjelenítéseknél ugyanúgy néznek ki, mégis logikailag különböző.
 - Három különböző karakter, ami nagyon hasonló megjelenésű:
 - a latin nagy „A”
 - a cirill nagy „А”
 - görög nagybetűs Alfa „Α”
- Példa
 - EXCLAMATION !
 - QUESTION_MARK ?
 - SEMICOLON ;

Karakterkódok – táblázatok

A karakterkód egy leképezés:

- Kölcsönösen egyértelmű megfeleltetést ad a karakterkészlet karakterei és a nemnegatív egészek egy halmaza között
- Egy egyedi számkódot, egy kódpozíciót rendel a karakterkészlet minden karakteréhez.
 - Gyakran táblázat
 - <http://unicode-table.com>

ASCII táblázat

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	\$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Karakterkódolás

A karakterkódolás egy *algoritmus* karakterek digitális formában történő megadására.

- Ez a karakterek kódszámainak sorozatait „oktetek” sorozatára képezi le.
- Például az ISO 10646 karakterkódban az 'a' az 'ä' és az %₀ (ezrelék jel) karaktereknek megfelelő számkódok a 97, a 228 és a 8240.

A karakterkódolás adja meg, hogy a karakterkódokat *hogyan* képezzük le oktetek sorozatára.

- Az ISO 10646 egy lehetséges kódolásában minden karakter kódolásához két oktetet használva a kódolási algoritmus ezeknek a (0, 97), a (0, 228) és a (32,48) oktetpárokat felelteti meg.

Milyen karakterek használhatók egy nyelvben?

Példák:

- Pascal (ISO 7185:1990):
 - Minimum követelmények
 - '0'..'9' rendezett és folytonos kell legyen
 - 'A'..'Z' rendezett, de nem feltétlenül folytonos
 - 'a'..'z' nem kötelező, de ha van, akkor mint a nagybetűk.
 - Nincs előírva a lehetséges karakterek halmaza
 - A különböző implementációk különböző kódolást használhatnak(!)
 - pl. EBCDIC vagy ASCII => különböző implementációk másképpen viselkedhetnek
 - '}' ∈ 'A'..'Z' EBCDIC-ben és '}' ∉ 'A'..'Z' ASCII-ben
 - 'A' < '1' EBCDIC-ben és '1' < 'A' ASCII-ben
 - Ez nem tesz jót a programok hordozhatóságának

Milyen karakterek használhatók egy nyelvben?

Példák:

- C++ - ISO/IEC 14882:2011 (és 2014 is)
 - Az alap karakterkészlet 96 elemű
 - a space
 - a horizontal tab
 - vertical tab
 - new-line
 - form feed vezérlő karakterei
 - A következő 91 grafikus karakter:
 - 'a'..'z' 'A'..'Z' '0'..'9'
 - _ { } [] # () < > % : ; . ? * + - / ^ & | ~ ! = , \ " ' ,
 - ISO 10646 szerint kódolva
 - Ez precízebb, mint az 1990-es, ahol nem volt a kód előírva.

Milyen karakterek használhatók egy nyelvben?

Példák:

- CLU, Eiffel:
 - ASCII kód a megengedett.
- ADA (ISO/IEC 8652:2012(E)):
 - A teljes ISO/IEC 10646:2011
- Java:
 - "Java programs are written using the Unicode character set"
- C#:
 - „A source file is an ordered sequence of Unicode characters.”
- Python:
 - „Python reads program text as Unicode code points; ...”

Elhatároló jelek

A legtöbb programozási nyelvben a helyköz (space), a tab, a sorvége elhatároló jel.

- Van sok egy karakterből álló elhatároló jel
 - Pascal: & ' () + - * / : . ; < = >
 - C++: & % ^ ' () + - { } | ~ [] \ " * / : . ; , < = > ! ?
 - Java: () { } [] ; , . + - * / : < = > ! ~ ? : & | ^ %
 - ADA: & ' () * + , - . / : ; < = > |
- Van számos több karakterből álló elhatároló jel is
 - Pascal: := >= <= <> << ** ..
 - C++: -> ++ -- .* ->* == <= >= && || << >> <<= >>= != .. +=
-= &= ^= |= :: *= /=
 - Java: == <= >= << != && || ++ -- >> >>> += -= *= /= &= |=
^= %= <<= >>= >>>=
 - ADA: => .. ** := /= >= <= << >> <> --

Megkülönbözteti-e a nyelv a kis- és nagybetűket?

Kutya = kutya = KUTYA = KuTya = ...?

Alapvetően két megközelítés

- Pascal, Fortran, CLU, ADA, ... esetén ekvivalensek
- C, C++, Modula, Perl, Java, C#, ... esetén minden betű különböző

De!

- Eiffel: a kis- és a nagybetűk ekvivalensek, de használhatjuk ugyanazt a nevet egy objektumra és a típusára
 - a: A
 - (itt az A az típus és a az objektum)

Azonosítók

Milyen karakterek használhatóak az azonosítók leírására?

Mi az azonosító megengedett szintaxisa?

Van-e hosszúsági megkötés az azonosítókra?

Vannak-e kötött szavak?

Van-e különbség a kulcsszavak és az előre definiált szavak között?

Milyen karakterek használhatóak azonosítókban?

Pascal (ISO 7185:1990):

- betűk ('A'..'Z', 'a'..'z') és számjegyek ('0'..'9')

C++(ISO/IEC 14882:2003), CLU, Eiffel:

- betűk ('A'..'Z', 'a'..'z'), számjegyek ('0'..'9') és '_'

Perl:

- az előzőeken túl \$, @ és % jellel is kezdődhetnek
 - \$ a skalárokat, @ számmal indexelt tömböt, % asszociatív tömböt jelöl.

ADA

- Ada2012 szabvány : minden, aminek a nevében szerepel, hogy betű vagy számjegy! (Πλάτων, Чайковский stb. is)

Java:

- A Unicode betűi és számjegyei, a '\$' és '_'

Milyen karakterek használhatóak azonosítókban?

C#:

- Az azonosítókat a Unicode szabvány ajánlásának megfelelően kell írni.
- Érdekesség, hogy a kulcsszavak a @ bevezető jellel használhatóak azonosítóként is (például: @bool)
 - Ez a jel más azonosítót nem vezethet be.
- Ezt a lehetőséget a programok hordozhatósága illetve "átjárhatósága" miatt vezették be.
- A @ prefixű szimbólumok neve valójában @ nélkül kerül fordításra.
 - Így attól függetlenül, hogy egy másik nyelv nem tartalmazza például a 'sealed' kulcsszót, és egy programban azonosítóként használják, C#-ban lehetőség van az azonosító közvetlen használatára.

Mi az azonosítók megengedett szintaxisa?

A leggyakrabban:

- letter { letter | digit }
- néha az '_' beszúrása is megengedett.

Pascal: letter{letter|digit}

ADA: letter{[_]letter|[_]digit},

- itt a betűk halmaza nagyobb, minden, aminek a nevében szerepel az, hogy „letter”

Java: Java_letter{[_]Java_letter|[_]digit},

- αβγδε vagy El_Niño

Van-e hosszúsági megkötés az azonosítókra?

Érdekesség

- Fortran66, Fortran77: maximum. 6 jel
- Fortran90: maximum. 31 jel

Általában tetszőleges hosszúság de, vannak kivételek

- ADA: be kell férjen egy sorba

Vannak-e „fenntartott” szavak?

Pascal, C, C++, Java, Perl, CLU, ADA, ...

- a fenntartott szavak (kulcsszavak) nem használhatók azonosítóként

ADA:

- különbséget tesz a kulcsszavak és az előredefiniált szavak között, mint pl. : Integer, True, etc..
- Az előredefiniált szavak átdefiniálhatóak:
 - `type Integer is range -999_999..+999_999;`
 - így a program implementáció-független
 - de: `True: Integer ...` -- nincs értelme...

Fortran, PL/1:

- megengedett a kulcsszavakat azonosítóként használni!

Literálok

Milyen numerikus literálok vannak?

Milyen más alapok megengedettek a 10-esen kívül?

Mi az egész, illetve valós literálok szintaxisa?

Többsoros sztring literálok megengedettek-e?

Vezérlőkértékek sztring literálokban megengedettek-e, és hogyan írjuk le őket?

Numerikus literálok

A gyakorlatban csak a '0'..'9' és az 'A'..'F' karakterek használhatók

- További lehetőségek:
 - '_' : 456_789
 - ADA, Perl, Eiffel
 - Exponenciális alak: 1E6
 - Ada
 - 'L' 'U' a típus megadására (long, unsigned): 4L
 - C, C++, Java, C#

Mi a számok szerkezete?

- Egészek: decimális: [-]digit{digit}
 - -123, 456789

Numerikus literálok

Mi a számok szerkezete?

- Racionális: `[-]digit{digit}.digit{digit}[exponent]`
 - `-123.456E+3`
- Majdnem minden nyelv ad ehhez valami specialitást:
 - Pascal
 - `[-]digit{digit}[exponent]` `12E+3`
 - ADA
 - `'_'` is lehet: `-1_234.0`
 - Eiffel
 - `[-]{digit}.{digit}[exponent]` `-1.`
 - Java
 - `"float_suffix"` is lehet a végén

Milyen alapok megengedettek?

Általában: decimális számok, néha más is

- Például 2, 8 16

Példák

- Pascal, Modula, Oberon: csak decimális számok
- C, C++, Java...: Oktális, decimális és hexadecimális
 - oktális: 0734
 - hexadecimális: 0xFF, 0xC0B0L.
- ADA: minden alap 2 és 16 között:
 - `base#mantissa#[exponent]`: 16#FFF.F#E1
- Mathematica: minden alap 2 és 36 között!
 - Minden számjegy és angol betű
 - $\text{basis}^{\text{mantissa}}$: $30^{\text{Mathematica}}$ = 13 207 019 439 499 57010

Karakterek és karaktersorozatok

Karakterek: 'A' vagy „escape szekvenciák”: '\n', '\', '\ooo' stb.

Sztringek "A"

Tárolásuk különböző lehet.

Megjegyzések

Szerkezete befolyásolja a program megbízhatóságát

Szokásos lehetőségek:

- egy speciális oszloptól (jeltől) a sor végéig
- speciális jel(ek) elején - végén
- speciális jel(ek) elején - vége a sor vége

Megjegyzések

Példák

- Pascal:
 - (* és *)
 - { és }
- CLU, Matlab
 - %-tól sor végéig
- Python
 - #-tól a sor végéig
 - " és " között
- C:
 - /* és */
- C++: mint C-ben, és még
 - // -tól sor végéig
- Java: mint C++-ban és még
 - /** documentation */
- C#: mint C++-ban is még
 - /// egysoros dokumentációs megjegyzés
- ADA, Eiffel:
 - -- -tól sor végéig

Beépített adattípusok, változók, kifejezések

Mit jelent a nyelvben az adattípus?

Egy értékthalmaz és egy művelethalmaz

Specifikáció – Reprezentáció – Implementáció

Példa

- Specifikáció
 - egészek $-\infty$ -től $+\infty$ -ig, és a megengedett műveletek $+$, $-$, $*$, stb.
- Reprezentáció
 - 8, 16, 32, 64-bit, előjeles kettes komplementes kóddal (megszorítások)
- Implementáció
 - a műveletek megvalósítása

Típus a programozó szemszögéből

Típusspecifikáció („mit”)

- alaphalmaz: a valós világ minket érdeklő objektumainak halmaza
- specifikációs invariáns (I_S) – ezt a halmazt tovább szűkíti
- típusműveletek specifikációja – csak a „mit”!

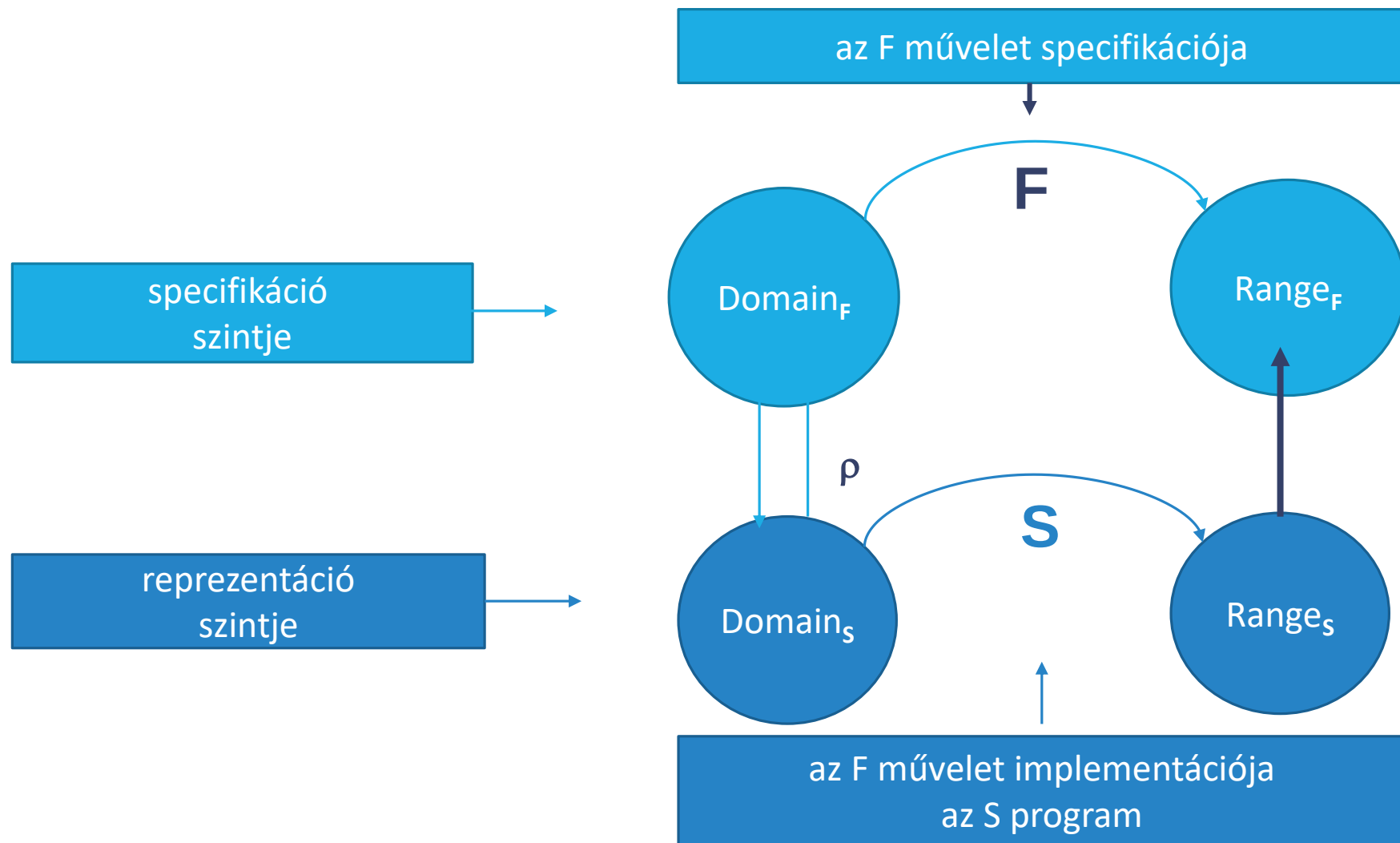
Típus a programozó szemszögéből

Típus megvalósítás („hogyan”)

- Reprezentációs függvény
- Típus invariáns
- Típusműveletek megvalósítása

Pl. komplex számok, verem, stb.

A típus specifikáció és a típus kapcsolata



Típusok támogatása

A legtöbb programozási nyelvben van valamilyen adattípusra támogatás.

Vannak „beépített” ("built-in") típusok, itt

- a specifikáció – a programozási nyelv referencia kézikönyvében
- a reprezentáció és az implementáció a fordítóprogrammal jön.
- Fontos: mennyire szabványosak a beépített típusok?

Vannak megengedett típuskonstrukciók

- tömb, rekord, stb.

A programozási nyelvek gyakran lehetővé teszik új adattípusok tervezésekor, hogy önálló modulokban elrejtsük a reprezentációt és az implementációt

A programozási nyelvek típusszerkezete

Hogyan osztályozhatjuk a típusokat?

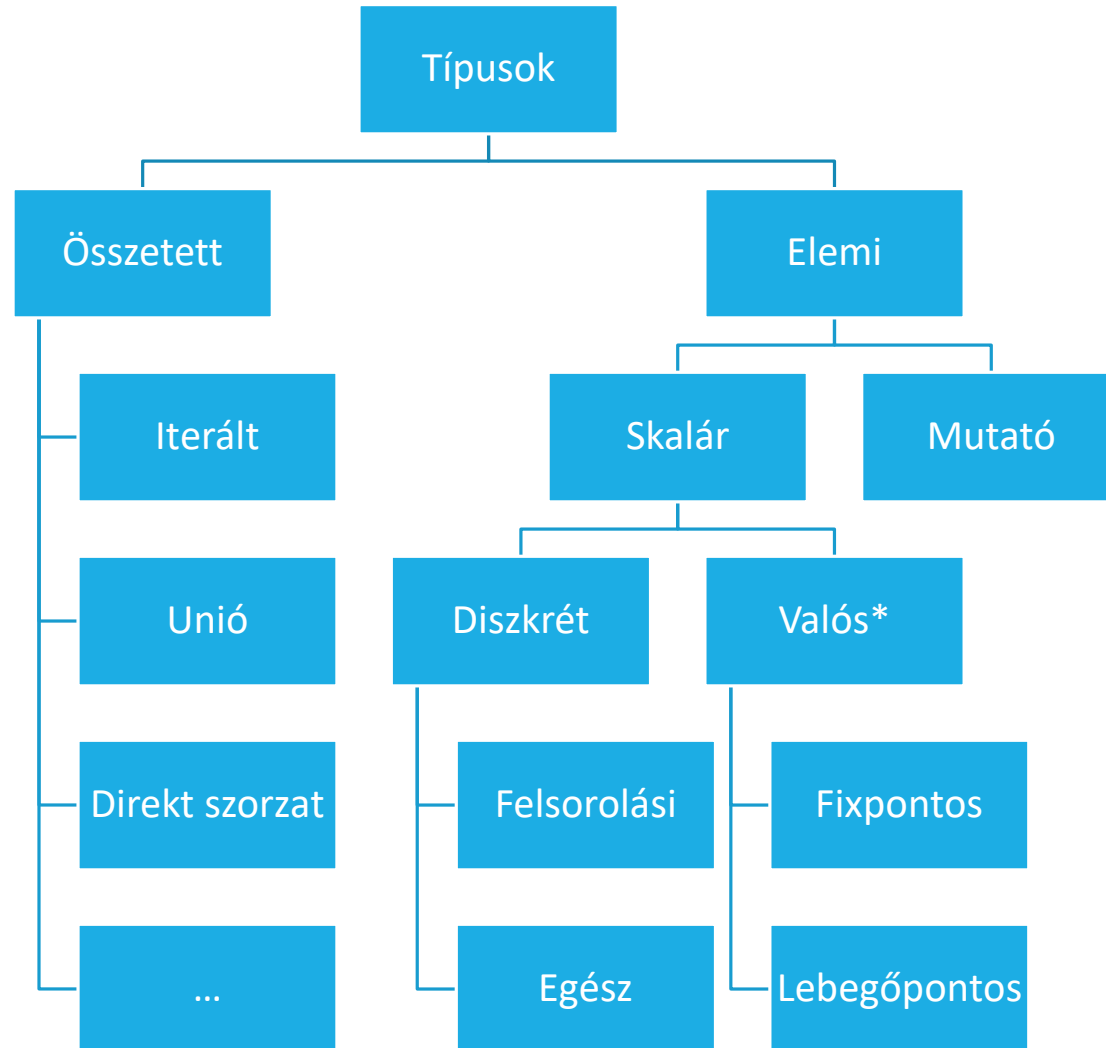
Vannak típusosztályok a nyelvben?

- Vannak típusosztályokra vonatkozó műveletek?

Elemi és összetett típus

- **Elemi** típusoknak nevezzük azokat a típusokat, amelyek logikailag felbonthatatlanok,
- Az **összetett** típusokat már meglévő típusokból, mint komponensekből hozzuk létre.

Típusosztályok



Típusosztály műveletekre példa

Ada: - felsorolási típus

- `type Hónapok is (Január, Február, Március,..., December);`
- automatikusan keletkezik:
 - `Hónapok'First`, `Hónapok'Last`,
 - `Hónapok'Range`,
 - `Hónapok'Min(x,y)`, `Hónapok'Max(x,y)`
 - `Hónapok'Succ(x)`, `Hónapok'Pred(x)`
 - `Hónapok'Image(x)`
 - `Hónapok'Value(x)`
 - `Hónapok'Val(x)`
 - `Hónapok'Pos(x)`
 - ...

Van-e speciális tulajdonsága a nyelv típusrendszerének?

ADA:

- Különbség az altípus és a származtatott típus között:
 - Egy típus altípusa a típus értékalmazának részalmazát jelöli.
 - Az altípusra alkalmazható a típus összes művelete.
- Például:
 - Dátum - a hónap napjai az 1 .. 31 intervallumba esnek.
 - Ehhez deklarálhatunk egy altípust:
 - `subtype Napok is Integer range 1 .. 31;`

Van-e speciális tulajdonsága a nyelv típusrendszerének?

ADA:

- Új típus létrehozása már létező típusból: származtatás
 - itt kicsit mást jelent, mint az OOP-ben
 - `type Hosszúság is new Float;`
 - A származtatott típus átveszi az eredeti típus struktúráját, értékalmazát és műveleteit, átveszi a kezdeti értékeit is.
 - DE: a két típus nem ekvivalens!
 - `type Terület is new Float;`
 - `H: Hosszúság; T: Terület;`
 - `H := T;` hibás! (már szintaktikailag!)
 - Konverzió lehetősége szükséges:
 - `function ”*(x, y: Hosszúság) return Terület is`
 - `begin return Terület (Float(x) * Float(y)); end;`
 - `T:=H*H;` írható, de `T:=H;` vagy `T:=H/H;` továbbra sem!

Van-e speciális tulajdonsága a nyelv típusrendszerének?

Java:

- Kétféle típus létezik: a primitív típusok (numerikus és logikai), és a referencia típusok.
- A felhasználó által definiált típusok az osztályok.
- A primitív típusoknak van „csomagoló” osztálya, boxing, unboxing lehetőségekkel

Eiffel:

- alapértelmezés szerint az értékek objektumokra vonatkozó referenciák, de definiálhatunk kiterjesztett (expanded) típusokat is, ezek változói maguk az objektumok.
- Az alap típusok mindig kiterjesztettek.
- Új típus = új osztály létrehozása.

C#: érték és referencia típusok.

- Érték típusok: egyszerű típusok (pl. char, int, float), felsorolási és struct típusok.
- Referencia típusok - osztály (class), interface típusok, delegate és tömb típusok.
- Boxing – unboxing lehetőségek vannak itt is.

Mik a beépített adattípusok?

Az alaptípusok általában, a szokásos műveletekkel:

- az egészek
- a karakterek
- a logikai típus
- a valósak

A skalár típusok osztálya

A skalár típusok a diszkrét és a valós típusok.

- Értékei rendezettek, így a relációs operátorok (<, <=, =, >=, >) előredefiniáltak ezekre a típusokra.
- ADA–ban számos attribútum:
 - S'First, S'Last, S'Range
 - S'Min, S'Max
 - S'Succ, S'Pred (Constraint_Error)
 - S'Image , S'Width, S'Value stb.
 - Hónapok'First = Január
 - Hónapok'Succ(Március)= Április

A skalár típusok osztálya

Eiffel –ben ezek a típusok mindig kiterjesztettek.

C++ - ezek az „integral” típusok

Perl

- Skalár adattípus
 - a neve kötelezően \$ jellel kezdődik
 - számok, stringek (!) és referenciák tárolására, a típusok közötti (numerikus, string,...) konverzió automatikus
 - `$skalar1 = 'string'; # karakterlanc`
 - `$skalar2 = 1234; # integer`
 - `$skalar3 = 4.5; # float`
 - `$skalar4 = $skalar3; # ertekadas`
 - `$skalar5 = \ $skalar2; # referencia`

A diszkrét típusok osztálya

A diszkrét típusok a

- felsorolási és az
- egész típusok.

Felsorolási típusok

A típusérték-halmaz megadható egy explicit felsorolással

- `type Days is (Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday);`

Gyakran a **karakter** és a **logikai** típus is előredefiniált felsorolási típus.

- A logikai értékek kezelésére szolgáló típust sok programozási nyelvben Boolean-nak hívják, két lehetséges értéke van, a True és a False, a szokásos műveletek a not, and, or és a xor.
- De vannak kivételek: például Yes, No Objective-C esetén

Felsorolási típusok

Object Pascal

- Egy felsorolási típus definíciója:
- `type Flower = (Rose, Tulip, Violet, Geranium);`
- Az azonosítóknak különbözőeknek, és a programban egyedieknek kell lenniük.
- Standard függvények
 - `Pred(X)`
 - `Succ(X)`
 - `Dec(X)`
 - `Inc(X)`
 - `<`
 - `...`

Felsorolási típusok

Object Pascal

- Logikai

- Boolean, ByteBool 1 byte
- WordBool 2 byte
- LongBool 4 byte
- ByteBool, WordBool, LongBool : numerikus boolean típusok, (minden numerikus értéket logikaiként értelmez) a nemnulla a True
 - -1 true, 0 false

- Karakter típusok

- ANSIChar (1 byte),
WideChar (2 byte, Unicode)

Felsorolási típusok

ADA

- Minden felsorolási literál a felsorolási típus egy önálló típusértéke, és van egy pozíció száma.
 - Az első felsorolási literál pozíció száma 0.
 - A rendezési relációt a felsorolás sorrendje adja. Egy felsorolási típus értékei lehetnek azonosítók vagy karakterek.
 - `type Color is (Red, Blue, Green);`
 - `type Roman_Digit is ('I', 'V', 'X', 'L', 'C', 'D', 'M');`
- Megengedett a felsorolási nevek átlapolása, és ha szükséges, a típusnév segít a megkülönböztetésben: `Color' (Red)`.

Felsorolási típusok

ADA

- Három előredefiniált karakter típus van
 - Character – ennek az értékei az ISO 10646 Basic Multilingual Plane (BMP) Row 00 256 kódpozíciójának (Latin-1)
 - Wide_Character típus értékei pedig az ISO 10646 Basic Multilingual Plane (BMP) 65536 kódpozíciójának
 - Wide_Wide_Character típus értékei pedig az ISO/IEC 10646:2011 2 147 483 648 kódpozíciójának felelnek meg.
- Logikai
 - `type Boolean is (False, True);`
 - speciális előredefiniált típus a szokásos műveleteken kívül az
 - "and then" és az
 - "or else" műveletekkel (lusta kiértékelés).

Felsorolási típusok

C++

- Az "enum" kulcsszót használják a felsorolási típusok, a megfelelő egész érték 0-val kezd, egyesével nő, kivéve, ha "= expr" szerepel valahol (!).
 - `enum color {red, blue, green=20, yellow};`
- Az előredefiniált `char`, `signed char` és `unsigned char` típusok írják le a lehetséges karakter típusokat, sok probléma származik abból, hogy ezeket nem szabványosították igazán.
- Előredefiniált logikai típus a `bool` a `false` és `true` értékekkel, kifejezésekben a 0 értéket is hamis-ként kezeli, és minden nem 0-t igaznak.

Felsorolási típusok

Java

- Nem volt felsorolási típus – a Java 5.0-ig!
- Az előredefiniált char típus a 16 bites Unicode character-készletet támogatja '\u0000' –tól '\uffff'-ig, azaz 0-tól 65535-ig.
- Előredefiniált logikai típus, a boolean a false és true értékekkel
- A műveletek
 - Relációs: == !=
 - not: !
 - and, or: & ||
 - and, or, xor (bitwise): & | ^
 - feltételes kifejezés: ? :

Felsorolási típusok

Java

- `enum Season { WINTER, SPRING, SUMMER, FALL }`
- Lehet állapota is és műveletei is
 - <https://docs.oracle.com/javase/tutorial/java/javaOO/enum.html>
 - Minden enum típusnak ez a (rejtett) közös őse
 - `public abstract class Enum<E> extends Enum<E>>
 extends Object
 implements Comparable<E>, Serializable`

Példa

```
import java.util.*;
```

```
public class Card {  
    public enum Rank { DEUCE, THREE, FOUR, FIVE, SIX,  
                      SEVEN, EIGHT, NINE, TEN, JACK, QUEEN, KING, ACE }
```

```
    public enum Suit { CLUBS, DIAMONDS, HEARTS, SPADES }
```

```
    private final Rank rank;  
    private final Suit suit;  
    private Card(Rank rank, Suit suit) {  
        this.rank = rank;  
        this.suit = suit;  
    }
```

```
    public Rank rank() { return rank; }  
    public Suit suit() { return suit; }
```

```
    public String toString() { return rank + " of " + suit; }
```

```
    private static final List<Card> protoDeck = new ArrayList<Card>();  
    // Initialize prototype deck
```

```
    static {  
        for (Suit suit : Suit.values())  
            for (Rank rank : Rank.values())  
                protoDeck.add(new Card(rank, suit));  
    }
```

Használja a Rank és a
Suit toString-jét

Használja a Rank és a
Suit values műveletét

Tulajdonságok és viselkedés

```
public enum Planet {  
    MERCURY (3.303e+23, 2.4397e6),  
    VENUS   (4.869e+24, 6.0518e6),  
    EARTH   (5.976e+24, 6.37814e6),  
    MARS    (6.421e+23, 3.3972e6),  
    JUPITER (1.9e+27, 7.1492e7),  
    SATURN  (5.688e+26, 6.0268e7),  
    URANUS  (8.686e+25, 2.5559e7),  
    NEPTUNE (1.024e+26, 2.4746e7),  
    PLUTO   (1.27e+22, 1.137e6);  
  
    private final double mass;    // in kilograms  
    private final double radius; // in meters  
    private Planet(double mass, double radius) {  
        this.mass = mass;  
        this.radius = radius;  
    }  
    private double mass() { return mass; }  
    private double radius() { return radius; }
```


Tulajdonságok és viselkedés – Absztrakt művelet

```
public enum Operation {  
    PLUS    { double eval(double x, double y) { return x + y; } },  
    MINUS   { double eval(double x, double y) { return x - y; } },  
    TIMES   { double eval(double x, double y) { return x * y; } },  
    DIVIDE  { double eval(double x, double y) { return x / y; } };  
    // Do arithmetic op represented by this constant  
    abstract double eval(double x, double y);  
}
```

Próba

```
public static void main(String args[]) {  
    double x = Double.parseDouble(args[0]);  
    double y = Double.parseDouble(args[1]);  
    for (Operation op : Operation.values())  
        System.out.printf("%f %s %f = %f%n", x, op, y, op.eval(x, y));  
}
```

Felsorolási típusok

C# - Enum

- Rendelkezik alaptípussal – ez tetszőleges integral típus lehet
 - byte, sbyte, short, ushort, int, uint, long, ulong
- Deklaráció példa:
 - ```
enum Color: long {
 Red,
 Green,
 Blue
}
```
- Alapértelmezésben int
  - hozzárendelt egész-értékek 0-val kezdődnek, ez átállítható
- System.Enum a közös őssztály

# Felsorolási típusok

---

## C# logikai típus:

- `bool` a `true` és `false` értékekkel, szokásos műveletek
- Nincs szabványos konverziós lehetőség!

## C# karakterek:

- `char` típus unsigned 16-bit egészek 0 és 65535 között.
- A Unicode karakter-halmaznak felel meg.
- Bár ugyanaz a reprezentációja, mint az `ushort`-nak, nem ugyanaz a művelethalmaz!
- A konstansokat kétféle módon lehet írni:  
karakter-literálként vagy egész literálként, explicit típuskényszerítéssel: `(char)10` ugyanaz, mint `'\x000A'`.

# Felsorolási típusok

---

## SWIFT

- Alaptípus megadása
- Hozzárendelt reprezentációs érték megadható
  - Értékek lekérdezhetők
- Példa

```
enum Rank: Int {
 case Ace = 1
 case Two, Three, Four, Five, Six, Seven, Eight, Nine, Ten
 case Jack, Queen, King
}

let ace = Rank.Ace
let aceRawValue = ace.rawValue
```

# Egész típusok

---

Az egész típusok nagyon közel vannak a számítógépes reprezentációhoz.

A műveletek általában a szokásosak, néhol gond van az osztással és a hatványozással

# Egész típusok

---

## Object Pascal lehetőségek:

- ShortInt signed 8 bit
- SmallInt signed 16 bit (csak Delphi 2, 3)
- Integer signed system-dep. Delphi 1 – 16 bit  
Delphi 2-től – 32 bit
- LongInt signed 32 bit
- Byte unsigned 8 bit
- Word unsigned 16 bit
- Cardinal unsigned system-dep. Delphi 1 – 16 bit  
Delphi 2-től – 32 bit

# Egész típusok

---

## ADA:

- Előjeles egészek:
  - `type Page_Num is range 1 .. 2_000;`
  - `type Line_Size is range 1 .. Max_Line_Size;`
- Maradékosztályok:
  - `type Byte is mod 256; -- egy előjel nélküli byte`
  - `type Hash_Index is mod 97;`
- Minden egész típust úgy tekintenek, mint ami a névtelen, előredefiniált `root_integer` típusból lett származtatva.
- Egész literálok ennek az `universal_integer` osztályába tartoznak.
  - Ez a szigorú típusosság miatt fontos.

# Egész típusok

---

## ADA:

- Az előredefiniált egész típus az Integer, van két előredefiniált altípusa:
  - `subtype Natural is Integer range 0 .. Integer'Last;`
  - `subtype Positive is Integer range 1 .. Integer'Last;`
- Az értékintervallumnak egy tetszőleges implementáció esetén tartalmaznia kell a  $-2^{15}+1 \dots +2^{15}-1$ -t.
- Megengedett, de nincs előírva:
  - `Short_Integer`
  - `Long_Integer`
  - `Short_Short_Integer`
  - `Long_Long_Integer`
  - ...



# Egész típusok

---

## ADA:

- A műveletek a szokásosak
  - minden attribútum,
  - relációs operátorok,
  - +, -, (unáris, bináris) \*, / (csonkít),
  - rem (maradék),
  - mod (modulus),
  - abs,
  - \*\* (hatványozás Natural kitevőre)
- A modulo típusok műveletei a maradékosztályokon

# Egész típusok

---

## C++:

- Az előredefiniált egész típusoknak 4 mérete lehet: `short int`, `int`, `long int`, `long long int`.
- "Longer integers provide no less storage than shorter ones."
- Az `unsigned int`, `unsigned long int`, `unsigned short int` típusokat modulo  $2^n$  aritmetikával használja (n a reprezentációban a bitek száma).
- A szabvány definiálja a minimum értékintervallumokat
- C++11 fix méretű típusok
  - `int8_t`, `int16_t`, `int32_t`, `int64_t`

# Egész típusok

## C++

- A műveletek:

|                      |                                   |
|----------------------|-----------------------------------|
| Relációs             | == , != < <= > >=                 |
| Unáris               | * + - &                           |
| Multiplikatív        | * / %                             |
| Additív              | + -                               |
| incr. prefix postfix | ++                                |
| decr. prefix postfix | --                                |
| shift előjeles       | << >>                             |
| Komplemens bitenként | ~                                 |
| Feltételes op.       | ? :                               |
| Sizeof               |                                   |
| pointer_to_member    | ->* .*                            |
| Értékadó op.         | = *= /= %= += -= >>= <<= &= ^= != |

# Egész típusok

---

## Java

- Az előredefiniált egész típusoknak előírt specifikációja van:

|       |                                             |
|-------|---------------------------------------------|
| byte  | -128..127                                   |
| short | -32768..32767                               |
| int   | -2147483648.. 2147483647                    |
| long  | -9223372036854775808 .. 9223372036854775807 |
| char  | '\u0000'..' \uffff', vagyis: 0..65535       |

# Egész típusok

## Java

- Operátorok

|                          |                   |
|--------------------------|-------------------|
| Relációs                 | == , != < <= > >= |
| Unáris                   | + -               |
| Multiplikatív            | * / %             |
| Additív                  | + -               |
| incr. Prefix postfix     | ++                |
| decr. Prefix postfix     | --                |
| shift előjeles, előjeln. | << >> >>>         |
| komplement bitenként     | ~                 |
| feltételes op.           | ? :               |

- További hasznos előredefiniált műveletek a csomagoló osztályokban

# Egész típusok

---

## C#

- Előírt specifikáció van itt is

|          |                 |                                              |
|----------|-----------------|----------------------------------------------|
| ◦ sbyte  | signed 8-bit    | -128 ... 127                                 |
| ◦ byte   | unsigned 8-bit  | 0 ... 255                                    |
| ◦ short  | signed 16-bit   | -32768 ... 32767                             |
| ◦ ushort | unsigned 16-bit | 0 ... 65535                                  |
| ◦ int    | signed 32-bit   | -2147483648 ... 2147483647                   |
| ◦ uint   | unsigned 32-bit | 0 ... 4294967295                             |
| ◦ long   | signed 64-bit   | -9223372036854775808 ... 9223372036854775807 |
| ◦ ulong  | unsigned 64-bit | 0 ... 18446744073709551615.                  |

- Műveleteknél előírták a konverziókat is.

# Egész típusok

---

## Eiffel:

- Az INTEGER kiterjesztett, a COMPARABLE és a NUMERIC osztályok leszármazottja.
- A reprezentáció legalább Integer\_bits bitet használ, ez egy platform-függő konstans, amelyet a PLATFORM osztály definiál.
- A műveletek:
  - < <= > >= a COMPARABLE-ból,
  - + - \* / a NUMERIC-ból,
  - és az újak:
    - hatványozás ^
    - egész osztás //
    - maradék \\

# Valós típusok

A valós típusok a valós számok **közelítései**.

- a lebegőpontos típusok - relatív pontosság
- fixpontos típusok - abszolút pontosság

A legtöbb programozási nyelv támogatja az előjeles lebegőpontos típusokat, ahol 1 bit az előjel, és a szám formátuma:



- $\text{mantissa} * 10^{\text{exponent}}$ , ahol  $0 \leq \text{abs}(\text{mantissa}) < 10$ .

A kitevőnek is lehet előjele

- A kitevőt általában az 'E' betű jelöli.



# Valós típusok

## Pascal

- A nyelv lebegőpontos valósakat használ.

| Típus:   | Reprezentáció:                                |
|----------|-----------------------------------------------|
| Single   | 32 bit (1+23+8)<br>1.401E-45 .. 3.402E38      |
| Real     | 48 bit (1+39+8)<br>2.9E-39 .. 1.7E38          |
| Double   | 64 bit (1+52+11)<br>4.941E-324 .. 1.797E308   |
| Extended | 80 bit (1+64+15)<br>3.363E-4932 .. 1.189E4932 |
| Comp     | 64 bit (1+63)<br>-9.2E-18 .. 9.2E18           |

# Valós típusok

---

## Object Pascal:

- Single, Double, Extended
  - IEEE nemzetközi szabvány szerint.
- Új fixpontos valós típus 4 számjegy pontossággal:

|                 |                                                          |               |
|-----------------|----------------------------------------------------------|---------------|
| <b>Currency</b> | <b>-922337203685477.5808 ..<br/>922337203685477.5807</b> | <b>8 byte</b> |
|-----------------|----------------------------------------------------------|---------------|

- Műveletek a szokásosak

# Valós típusok

---

## ADA:

- A nyelv ad lehetőséget a lebegőpontos és a fixpontos típusok kezelésére, de csak egy előredefiniált lebegőpontos típus van, a Float.
- A lebegőpontos típusoknál a relatív pontosság, míg a fixpontos típusoknál az abszolút pontosság megadására van lehetőség. Megadható egy értéktartomány is:
  - `type Real is digits 8;`
  - `type Coefficient is digits 10 range -1.0 .. 1.0;`
  - `type Mass is digits 7 range 0.0 .. 1.0E35;`

# Valós típusok

---

## ADA:

- Minden valós típust úgy tekintenek, mint egy előre definiált `root_real` típusból származtatott típust. A valós literálok ennek az osztályába tartoznak, így `universal_real` típusúak.
- Ha egy implementációban a lebegőpontos típusok pontossága legalább 6 számjegy, akkor a `Float` típus pontossága is legalább ennyi kell legyen.
- Megengedett, hogy egy implementáció támogasson további előre definiált lebegőpontos típusokat, pl.: `Short_Float`, `Long_Float`, `Short_Short_Float`, `Long_Long_Float`...
- A szokásos műveletek, minden attribútum, relációs operátorok, `+`, `-`, (unary, binary) `*`, `/` `abs`, `**` (hatványozás).

# Valós típusok

---

C++:

- Az előre definiált valós típusok a `float`, `double` és a `long double`.
- "The type `double` provides no less precision than `float`, and the type `long double` provides no less precision than `double`., ...

# Valós típusok

C++:

- Műveletek

|                                      |                                   |
|--------------------------------------|-----------------------------------|
| egyenlő, nem egyenlő összehasonlítás | == , != < <= > >=                 |
| indirekció, előjel, cím              | * + - &                           |
| szorzás, osztás, moduló              | * / %                             |
| összeadás, kivonás                   | + -                               |
| növelés prefix/postfix               | ++                                |
| csökk. prefix/postfix                | --                                |
| léptetés balra/jobbra                | << >>                             |
| bitenkénti komplement                | ~                                 |
| feltételes kifejezés                 | ? :                               |
| obj/típus mérete                     | sizeof                            |
| tagkiválasztás                       | ->* .*                            |
| értékadások                          | = *= /= %= += -= >>= <<= &= ^= != |

# Valós típusok

---

## Java:

- Az előredefiniált lebegőpontos típusoknak előírt pontossága van

|        |                 |
|--------|-----------------|
| float  | 32 bit IEEE 754 |
| double | 64 bit IEEE 754 |

- A float típus  $s \cdot m \cdot 2^e$  alakú, ahol
  - $s = +1$  vagy  $-1$ ,
  - $m$  egy pozitív egész, kisebb, mint 224
  - $e$  egy egész a  $-149..104$  intervallumból.
- A double típus  $s \cdot m \cdot 2^e$  alakú, ahol
  - $s = +1$  vagy  $-1$ ,
  - $m$  egy pozitív egész, kisebb, mint 253
  - $e$  egy egész a  $-1075..970$  intervallumból

# Valós típusok

---

## Java:

- Érdekesség:
  - a pozitív és negatív végtelenek (POSITIVE\_INFINITY, NEGATIVE\_INFINITY)
  - speciális Not-a-Number (NaN) érték is szabvány.
    - Ha  $x$  a NaN mi az értéke az  $x \neq x$ -nek?
- Műveletek a NaN-ra nem értelmezettek
  - További hasznos műveletek a Float, Double és Math osztályokban.



# Valós típusok

## Java:

- Műveletek a NaN-ra nem értelmezettek
  - További hasznos műveletek a Float, Double és Math osztályokban.

|                                  |                   |
|----------------------------------|-------------------|
| Relációs                         | == , != < <= > >= |
| Unáris                           | +-                |
| Multiplikatív                    | * / %             |
| Additív                          | + -               |
| Incr. prefix, postfix            | ++                |
| Decr. prefix, postfix            | --                |
| Shift előjeles és előjel nélküli | << >> >>>         |
| Komplement bitenként             | ~                 |
| Feltételes op.                   | ? :               |

# Valós típusok

---

C#:

- Javához hasonló - 2 lebegőpontos típus: float és double

|        |                 |
|--------|-----------------|
| float  | 32 bit IEEE 754 |
| double | 64 bit IEEE 754 |

- Pozitív és negatív 0
  - legtöbbször ugyanaz, de osztásnál más lehet.
- Pozitív és negatív végtelen ( $\infty$ )
  - $1.0 / 0.0$  a pozitív végtelen
  - $-1.0 / 0.0$  a negatív.
- A Not-a-Number érték (NaN).
  - $0.0/0.0$

# Valós típusok

---

## C#

- Decimal típus a pénzügyi számításokhoz:
  - 128-bites adattípus, értékei:  $1.0 \times 10^{-28}$  ...  $7.9 \times 10^{+28}$ , 28-29 szignifikáns számjeggyel.
  - Lényeg, hogy a tízes számrendszerbeli törtértékeket is pontosan ábrázolja, műveleteknél szokásos módon kerekít.

# Valós típusok

---

## Eiffel

- A REAL osztály kiterjesztett, a COMPARABLE és a NUMERIC osztályok leszármazottja.
- A reprezentáció legalább Real\_bits bitet használ, ez egy platform-függő konstans, amelyet a PLATFORM osztály definiál.
- A műveletek
  - < <= > >= a COMPARABLE-ból,
  - + - \* / a NUMERIC-ból,
  - és az újak
    - hatványozás ^
    - stb.