

Beépített adattípusok, változók, kifejezések

- Áttekintés
- Mit jelent a nyelvben az adattípus?
- A programozási nyelvek típusstruktúrája
- Hogyan osztályozhatjuk a típusokat? Vannak típusosztályok a nyelvben?
- Van-e valamilyen speciális tulajdonsága a nyelv típusrendszerének?
- Mik a beépített adattípusok?

2005.02.21.

1

- Hogyan definiálhatunk új adattípusokat?
- Mik a megengedett típuskonstrukciók? Elérhetők-e az alábbi típuskonstrukciók, és ha igen, milyen tulajdonságokkal?
 - tömb típusok
 - rekord típusok
 - halmazok
- Mikor ekvivalens két típus?
 - Strukturális ekvivalencia
 - Név szerinti ekvivalencia
- Mikor és hogyan lehetséges típuskonverzió?

2005.02.21.

2

- Változók
 - Láthatóság, Elérhetőség
 - Hogyan definiálhatunk változókat és konstansokat?
- Kifejezések
 - Prefix - Postfix - Infix jelölés
 - A műveletek precedencia szintjei

2005.02.21.

3

Mit jelent a nyelvben az adattípus?

- Egy érték-halmaz és egy művelet-halmaz.
- specifikáció – reprezentáció – implementáció
 - pl. specifikáció: egészek $-\infty$ -tól $+\infty$ -ig, és a megengedett műveletek $+$, $-$, $*$, stb.
 - reprezentáció: 8, 16, 32, 64-bit, előjeles kettes komplement kódokkal (megszorítások)
 - implementáció: a műveletek megvalósítása

2005.02.21.

4

- A legtöbb programozási nyelvben van valamilyen adattípusra támogatás.
- Vannak ún. „beépített” („built-in”) típusok, itt
 - a specifikáció – a programozási nyelv referenciakönyvében
 - a reprezentáció és az implementáció a fordítóprogrammal jön.
- Fontos: mennyire szabványosak a beépített típusok??

2005.02.21.

5

- Vannak megengedett típuskonstrukciók
 - tömb, rekord, stb.
- A programozási nyelvek gyakran lehetővé teszik új adattípusok tervezésekor, hogy önálló modulokban elrejtjük a reprezentációt és az implementációt \Rightarrow jön.

2005.02.21.

6

A programozási nyelvek típus szerkezete

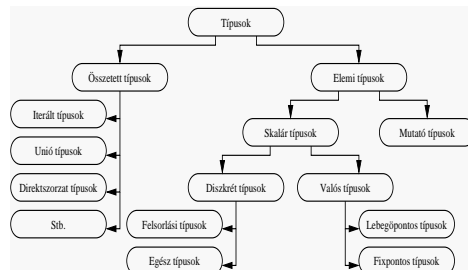
- Hogyan osztályozhatjuk a típusokat?
- Vannak típusosztályok a nyelvben? Vannak típusosztályokra vonatkozó műveletek?

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

2005.02.21.

7

Típusosztályok



2005.02.21.

8

- Műveletekre példa:

Ada95:

```
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) stb.
```

2005.02.21.

9

- Van-e valamilyen speciális tulajdonsága a nyelv típusrendszerének?*

ADA95:

- különbség az *altípus* és a *származtatott típus* között. Egy típus altípusa a típus értékhalmozának részhalmazát jelöli. Az altípusra alkalmazható a típus összes művelete,
- Például: Dátumok - a hónap napjai az 1 .. 31 intervallumba esnek.
Ehhez deklarálhatunk egy altípust:
subtype Napok is Integer range 1 .. 31;

Új típus létrehozása már létező típusból: származtatás

```
type Hosszúság is new Float;
```

- A származtatott típus átveszi az eredeti típus struktúráját, értékhalmozát 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!
```

- Konverzió lehetősége szükséges:

```
T := Terület (Float(H) * Float(H));
```

vagy:

```
function "**"(x, y: Hosszúság) return Terület is
begin return Terület (Float(x) * Float(y)); end;
```

definíciósával:

```
T:=H*H; írható, de T:=H; vagy T:=H/H; továbbra sem!
```

2005.02.21.

11

- Java:** kétféle típus létezik: a primitív típusok (numerikus és logikai), és referencia típusok. A felhasználó által definiált típusok az osztályok.
- Eiffel:** alapértelmezés szerint az értékek objektumokra vonatkozó *referenciák*, de definiálhatunk kiterjesztett (expanded) típusokat is, ezek változói maguk az objektumok. Az alap típusok is kiterjesztettek. Egy új típus definíciója mindig egy új osztály létrehozását jelenti.
- 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.

2005.02.21.

12

Mik a beépített adattípusok?

- Az alaptípusok általában
 - az egészek,
 - a karakterek,
 - a logikai típus, és
 - a valósak,
- a szokásos műveletekkel.

2005.02.21.

13

A skalár típusok osztálya

- A *skalár típusok* a *diszkrét* és a *valós típusok*.
- Értékei **rendezettek**, így a relációs operátorok (<, <=, =, >=, >) előredefiniáltak ezekre a típusokra.
- **ADA95** –ben 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

2005.02.21.

14

- **Eiffel** –ben ezek a típusok mindig kiterjesztettek.
 - C++ - ezek az ún „integral” típusok
 - Perl:
 - Skalár adattípus:
 - a neve kötelezően \$ jellel kezdődik
- számok, sztringek és referenciák tárolására, a típusok közötti (numerikus, string,...) konverzió automatikus
- ```
$skalar1 = 'string'; # karakterlanc
$skalar2 = 1234; # integer
$skalar3 = 4.5; # float
$skalar4 = $skalar3; # értékadás
$skalar5 = \ $skalar2; # referencia
```

2005.02.21.

15

## A diszkrét típusok osztálya

- A *diszkrét típusok* a *felsorolási* és az *egész* típusok.
- **ADA95** –ben minden diszkrét típusú értéknek van egy pozíciószáma, ami egy egész érték.  
Attribútumai: S'Pos, S'Val (Constraint\_Error)

2005.02.21.

16

## Felsorolási típusok

- A típusérték-halmaz megadható egy explicit felsorolással, mint pl.:  
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.

2005.02.21.

17

### 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), <, etc..
- A **Char** típus az ASCII szabványt írja le, 1 byte-ban.
- A karakterek megadhatók: 'A' vagy: #65, vagy mint vezérlő karakterek: ^L
- *Standard függvények*: Ord(C), Char(I), Upcase(C) etc.
- A **Boolean** egy speciális felsorolási típus, lehetséges értékei a False és a True, a szokásos műveletekkel.

2005.02.21.

18

### Object Pascal:

- Felsorolási – mint Pascalban
- Logikai:  
Boolean, ByteBool 1 byte  
WordBool 2 byte  
LongBool 4 byte  
ByteBool, WordBool, LongBool : numerikus  
boolean típusok, a nemnulla a True  
-1 true, 0 false
- Karakter típusok:  
ANSIChar (1 byte), WideChar (2 byte, Unicode)

2005.02.21.

Nyékyné Gaizler Judit

19

### ADA95:

- 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 megleltönböztetésben: Color' (Red) .

2005.02.21.

20

- Két 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 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).

2005.02.21.

21

C++:

- Az "enum" kulcsszót használják a felsorolási típusok, 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.

2005.02.21.

22

### Java:

- **Nem volt** felsorolási típus – a Java 5.0-ig!
- Az előredefiniált char típus a 16 bites Unicode character-készletet támogatja 'u0000' –től 'uffff'-ig, azaz 0-tól 65535-ig.
- Előredefiniált logikai típus, a boolean a false és true értékekkel, a műveletek:

|                        |         |
|------------------------|---------|
| reláció                | ==, !=  |
| not                    | !       |
| and, or                | &&,     |
| and, or, xor (bitwise) | &,  , ^ |
| feltételes kif.        | ? :     |

2005.02.21.

23

### • Java 5.0: Új enum típus!!

- Hasonlít a „hagyományos”-hoz:

```
enum Season { WINTER, SPRING, SUMMER, FALL }
```

- de új:

```
– van egy
public abstract class Enum<E extends
 Enum<E>>
 extends Object implements Comparable<E>,
 Serializable
```

- minden enum típusnak ez a közös őse
- lehet állapota is és műveletei is ([Enums Java.htm](#))

2005.02.21.

Nyékyné Gaizler Judit

24

- 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

- Példa (folyt.):  
Lehet olyan is, hogy tulajdonságokat és viselkedést adunk hozzá:

```
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
 Planet(double mass, double radius) {
 this.mass = mass;
 this.radius = radius;
 }
 private double mass() { return mass; }
 private double radius() { return radius; }
}
```

2005.02.21.

Nyékyné Gaizler Judit

26

- Példa (folyt.):  
Lehet olyan is, hogy tulajdonságokat és viselkedést adunk hozzá:

```
....
// universal gravitational constant (m3 kg-1 s-2)
public static final double G = 6.67300E-11;

double surfaceGravity() {
 return G * mass / (radius * radius);
}
double surfaceWeight(double otherMass) {
 return otherMass * surfaceGravity();
}
}

• ennek használata:
public static void main(String[] args) {
 double earthWeight = Double.parseDouble(args[0]);
 double mass = earthWeight/EARTH.surfaceGravity();
 for (Planet p : Planet.values())
 System.out.printf("Your weight on %s is %f%n",
 p, p.surfaceWeight(mass));
}
}
```

2005.02.21.

Nyékyné Gaizler Judit

27

- Lehet egyedi viselkedést is hozzáadni – absztrakt a művelet a típusban, és a konkrét megvalósítás az enum „objektumban” például:

```
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);
}

• Mintaprogram, ami kipróbálja:
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));
}
```

## Eiffel:

- **Nincs** felsorolási típus, *unique* deklarációkat használhatunk helyette:

```
Blue, Red, Green, Yellow:
INTEGER is unique;
```

- A BOOLEAN osztály az egyik alap-osztály, műveletei: and, and then, implies, not, or, or else és xor.

2005.02.21.

29

- C# felsorolási típus:

Rendelkezik alaptípussal – ez tetszőleges integral típus lehet byte, sbyte, short, ushort, int, uint, long, ulong

deklaráció:

attributumokopt módosítóokpt enum azonosító egész típus típus-törzs ;opt

Példa:

```
enum Color: long {
 Red,
 Green,
 Blue
}
```

- alapértelmezésben int
- hozzárendelt egész-értékek 0-val kezdődnek, ez átváltható.
- System.Enum a közös ősosztály

2005.02.21.

Nyékyné Gaizler Judit

30

- 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'.

2005.02.21.

Nyékyné Gaizler Judit

31

## Egész típusok

- Az egész típusok nagyon közel vannak a számítógépes reprezentációhoz. A műveletek általában a szokásosak, néhol gond van az osztással és a hatványozással.

2005.02.21.

32

- **Pascal:** a Turbo Pascal lehetőségeinek áttekintése:

| Típus:   | Reprezentáció:                         |
|----------|----------------------------------------|
| Byte     | 1byte, unsigned (0..255)               |
| ShortInt | 1byte, signed (-128..127)              |
| Word     | 2 byte, unsigned (0.. $2^{16}-1$ )     |
| Integer  | 2 byte, signed ( $-2^{15}..2^{15}-1$ ) |
| LongInt  | 4 byte, signed ( $-2^{31}..2^{31}-1$ ) |

2005.02.21.

33

- Műveletekre példa:

NOT (bitenként) NOT 0 = -1,  
NOT -15 = +14  
(NOT 11110001 = 00001110)

Negatív számok kettes komplementes ábrázolása:

1. vegyük a szám kettes számrendszerbeli alakját:  
 $15 = 8+4+2+1$       00001111
2. fordítsuk meg a biteket:      11110000
3. adjunk hozzá 1-t:      11110001

2005.02.21.

34

DIV                    123 DIV 4 = 30, 2 DIV 5 = 0  
MOD (remainder) 123 MOD 4 = 3, -12 MOD 5 = -2  
AND (bitenként) 12 AND 22 = 4  
                         (00001100 AND 00010110 = 00000100)  
OR (bitenként) 12 OR 22 = 30  
                         (00001100 OR 00010110 = 00011110)  
XOR (bitenként) 12 XOR 22 = 26  
                         (00001100 XOR 00010110 = 00011010)  
SHL (sh. L. bw) 2 SHL 7 = 256

2005.02.21.

35

- *Standard függvények:*

- Dec(i,n), Inc(i,n),
- Pred(I), Succ(I),
- Abs(I),
- Sqr(I),
- Round(R), Trunc(R),
- Hi(x), Lo(x) - felső/alsó byte

2005.02.21.

36

### Object Pascal lehetőségek:

|          |          |                                                        |
|----------|----------|--------------------------------------------------------|
| ShortInt | signed   | 8 bit                                                  |
| SmallInt | signed   | 16 bit (csak Delphi 2, 3)                              |
| Integer  | signed   | system-dep. Delphi 1 – 16 bit<br>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<br>Delphi 2-től – 32 bit |

2005.02.21.

Nyékyné Gaizler Judit

37

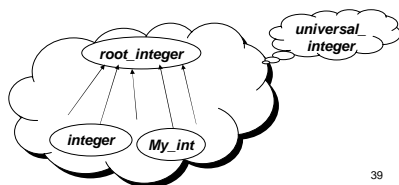
### ADA95:

- Előjeles egészek - Maradékosztályok  
type Page\_Num is range 1 .. 2\_000;  
type Line\_Size is range 1 .. Max\_Line\_Size;  
type Byte is mod 256; -- egy előjel nélküli byte  
type Hash\_Index is mod 97;
- Altípusokat is létrehozhatunk:  
subtype Small\_Int is Integer range -10 .. 10;  
subtype Column\_Ptr is Line\_Size range 1 .. 10;
- Egy egész érték pozíciószáma maga az érték.

2005.02.21.

38

- 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.



2005.02.21.

39

- 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 értéktartományuknak 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 stb. típusok.

2005.02.21.

40

- A műveletek a szokásosak, minden attribútum, relációs operátorok, +, -, (unáris, bináris) \*, / (csonkít), rem (maradék,  $A = (A/B)*B + (A \text{ rem } B)$ , ahol  $(A \text{ rem } B)$  előjele A-é), mod (modulus,  $A = B*N + (A \text{ mod } B)$ , ahol  $(A \text{ mod } B)$  előjele B-é), abs, \*\* (hatványozás Natural kitevőre).
- A moduló típusok műveletei a maradékosztályokon.

2005.02.21.

41

### C++:

- Az előredefiniált egész típusoknak 3 mérete lehet: short int, int, long int.
- "Longer integers provide no less storage than shorter ones." Egyébként a méretek implementáció-függőek.
- Az unsigned int, unsigned long int, unsigned short int típusokat modulo  $2^n$  aritmetikával használja (n a reprezentációban a bitek száma).

2005.02.21.

42

- A műveletek:

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

**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       |

2005.02.21.

44

- Operátorok:

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

- További hasznos előredefiniált műveletek a Byte, az Integer, a Long (és a Character) osztályokban.

- 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.

2005.02.21.

Nyékyné Gaizler Judit

46

## 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-ből,
  - és az újak:
    - hatványozás ^
    - egész osztás //
    - maradék \%

2005.02.21.

47

## 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: mantissa \* 10<sup>exponent</sup>, ahol 0 <= abs(mantissa) < 10.
- A kitevőnek is lehet előjele. A kitevőt általában az 'E' betű jelöli, és ha megengedett, hogy tizedesponet nélkül írjuk a számot, akkor általában kitevő rész szükséges.

2005.02.21.

48



**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           |

• Művelet:Példa:

\* 1E8\*1E5=1E13

/ (valós osztás) 123/4=30.75, 1/1E6=1E-6

• A valós osztás eredménye akkor is valós, ha az operandusok egészek.

• Standard függvények: Int(R) (egész rész, valós számot ad vissza), Abs(R), Sqr(R), Sqrt(R), Pi, Random(N), Sin(R) (R radiánban van),

• Ln(R), Exp(R) ( $e^R$ ) (a hatványozás helyett használhatjuk az Ln és Exp függvényeket - pl.:  $A^B = \text{Exp}(\text{Ln}(A)*B)$ )

2005.02.21.

50

### Object Pascal:

Single, Double, Extended az IEEE nemzetközi szabvány szerint.

Új fixpontos valós típus 4 számjegy pontossággal:

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

2005.02.21.

Nyékyné Gaizler Judit

51

### ADA95:

• 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 értékintervallum is. Pl.:

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;

2005.02.21.

52

- 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).

2005.02.21.

53

### C++:

• Az előre definiált valós típusok a float, double és a long double.

• "The type long provides no less precision than float, and the type long double provides no less precision than long." Különböző típusok implementáció-függőek.

2005.02.21.

54

• 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                       | -> *, *                           |
| értékkadások                         | = *= /= %= += -= >>= <<= &= ^= != |

**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 és  $e$  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 és  $e$  egy egész a -1075..970 intervallumból.

2005.02.21.

56

- **Érdekes:** a pozitív és negatív végtelenek (POSITIVE\_INFINITY, NEGATIVE\_INFINITY) és a speciális Not-a-Number (NaN) érték is szabvány. (Ha  $x$  a NaN mi az értéke az  $x != x$ -nek?)
- Műveletek (kivéve a NaN-t) (További hasznos műveletek a Float, Double és Math osztályokban.):

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

- 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 pl. osztásnál más lehet.
- Pozitív és negatív végtelen ( $\infty$ )  
Pl. 1.0 / 0.0 a pozitív végtelen -1.0 / 0.0 a negatív.
- A Not-a-Number érték (NaN). Pl. 0-val való osztásnál...
- A float típus  $s \cdot m \cdot 2^e$  alakú, ahol  $s$  +1 vagy -1,  $m$  egy pozitív egész, kisebb, mint 224 és  $e$  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 és  $e$  egy egész a -1075..970 intervallumból.
- A float típus értékei közelítőleg:  
 $1.5 \times 10^{-45} \dots 3.4 \times 10^{38}$ , 7 számjegy pontossággal.

2005.02.21.

Nyékyné Gaizler Judit

58

• C# - decimal típus a pénzügyi számításokhoz:

- 128-bites adat típus, értékei:  $1.0 \times 10^{-28} \dots 7.9 \times 10^{28}$ , 28-29 szignifikáns számjeggyel.
- Lényeg: a tízes számrendszerbeli törtértékeket is pontosan ábrázolja, műveleteknél szokásos módon kerekít.
- Nincs +0, -0, NaN.

2005.02.21.

Nyékyné Gaizler Judit

59

**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-ből,
  - és az újak:
    - hatványozás ^ stb.

2005.02.21.

60