PROGRAMOZÁSI NYELVEK - ADA. GYAKORLAT JEGYZET

Szerkesztette: *Balogh Tamás*

2013. október 31.

Ha hibát találsz, kérlek jelezd a *info@baloghtamas.hu* e-mail címen!

Előszó

A jegyzet nem teljes és nem hibátlan. Az esetlegesen előforduló hibákért, és kimaradt részekért felelőséget nem vállalok. A jegyzet megtanulása nem biztos, hogy elég a sikeres vizsgához.

Az előadás sorszáma a hetek sorszámával egyezik meg. A szünetek, ünnepnapok miatt elmaradt órákat is számítom.

1. Gyakorlat

Alap adattípusok

- *Integer:* `Type Integer is 1..128`
- *Boolean:* `Type Boolean is {TRUE, FALSE)`
- *Float*
- *Character*

Megszorított típus

Örökli a szülő típus típusműveleteit, struktúráját.

```
Type <név> Is New <szülőtípus> [Range n..m];
```

Altípusok

Positive: `Subtype Positive is Integer Range 1..Integer'Last;`

Típuskonverzió

Az ADA azt figyeli, hogy ugyanolyan típusú-e. Ezért erősen típusos. Nincs implicit típuskonverzió. Integerből származik a Positive és a Natural is. Kasztolással megoldható.

Attribútumok

A diszkrét típusok attribútumai:

- T'First – típusértékhalmaz alsó határa
- T'Last – típusértékhalmaz felső hazára
- T'Range – típusértékhalmaz egész intervalluma
- T'Min(a,b) – a,b közül a kisebb
- T'Max(a,b) – a,b közül a nagyobb
- T'succ(a) – az a érték rákövetkezője
- T'Pred(a) – az a érték megelőzője
- T'Image(a) – az a-t stringgél konvertálja
- T'Value(s) – az s stringből T típusú értéket konvertál
- T'(Width(a) – a T'Image által visszaadott string maximális hossza

Fordítás

gnatmake nev.adb - paranccsal fordítunk. Linker által generált file a nev.ali.
A linkelés befejeztével kapjuk meg a futtatható állományt.

Változó deklarálás

```
p : Positive = 12;
```

Egyéb

Commentezés a -- paranccsal történik. ADA-ban nem lehet logikai kifejezés helyére számot adni.

Műveletek és precedenciájuk

- **; abs; not
- *; /; mod; rem
- +; - (egyoperandusú)
- +; - (kétoperendusú)
- =; /=; <=; <; >; >=; in; not in
- and; or; yor; and then; or else

Vezérlési szerkezetek

Logikai kifejezések összekötése

Legyen a és b két logikai kifejezés. Ekkor a-t és b-t összeköthetjük:

- **and**: Ha a és b igaz, akkor igaz, különben hamis. Mindkét operandus kiértékelődik.
- **or**: Ha a és/vagy b igaz, akkor igaz, különben hamis. Mindkét operandus kiértékelődik.
- **and then** Ha a hamis, akkor hamis, ha a igaz, akkor b-től függ.
- **or else**Ha a igaz, akkor igaz, ha a hamis, akkor b-től függ.

Elágazás

```
If feltétel Then
  ...
Elsif feltétel2 Then
  ...
Else
  ...
End if;
```

Switch

```
Case feltétel is
  When érték => ...
  When érték => ...
  When Other => ...
  End Case;
```

Fentről lefelé vizsgálja az egyezőséget. A teljes típusérték halmazt le kell fedni.
Megendegett a | jellel egybe vonni a lehetőségeket: `When 'a'| 'b' | 'd' =>`

For Ciklus

```
For I in 1..10 Loop
  ...
End Loop;
```

Megadhatjuk a teljes típusérték halmazt: `Positive'First .. Positive'Last` ami megegyezik a következővel: `Positive'Range`
`Reverse` kulcsszóval hátulról előre járhatjuk be.

While Ciklus

```
While feltétel Loop
  ...
End Loop;
```

Végtelen Ciklus

```
Loop
  ...
End Loop;
```

Hátultesztelős Ciklus

```
Loop
  ...
Exit When feltétel;
End Loop;
```

2. Gyakorlat

Az ADA procedurális programozási nyelv. A függvény és az eljárás fogalma élesen elválik.

Alprogramok

Egy alprogram lehet eljárás(procedure), vagy függvény(function).
Eljárás megváltoztatja az állapotteret, míg a függvény csak számolást végez. A 2012-es ADA-ban bevezették, hogy a függvények is módoshíthassák az állapotteret.

```
function f(i : in out Integer) Return Integer
Begom
  i := i + 1; --megváltozik az i értéke, de a return 10-el tér vissza
  return 10;
End f;

i : Integer = 0;
i := f(i);
```

Eljárás

```
procedure <név>(...) -- egy eljárás deklarációja.
is
  --
Begin
  --
```

```
End<név>
```

Függvény

```
Function<név>(...) return T --T az a típus, amelynek típusérték halmazának
    valamely elemével térünk vissza
is
  --
Begin
  --
  return (T valamely eleme).
End<név>
```

Egy függvény nem állhat önmagában a kódban. Értékadás jobb oldalán, vagy valamilyen kiértékelés részeként(akár egy `if`-ben).
Nem változtatja meg az állapotteret.
Formális paraméter a függvény definiciójában lévő paraméter, aktuális paraméter pedig az, amivel meghívjuk a függvényt.

Paraméter átadás értékadásnak felel meg.

ADA-ban többféle paraméter átadás van:

- `in`: csak olvasni lehet.
- `out`: amíg nem írtuk, nem olvashatjuk
- `in out`: Megköveteljük, hogy legyen valamilyen értéke, értékadás bármely oldalán helyezkedhet.

```
Procedure P(i: in out Integer) is
..
Begin
..
End P
```

Függvénynek csak IN típusú paramétereik lehetnek.
Ha nem mondjuk meg explicit, hogy egy paraméter milyen, akkor az mindig IN típusú.
Paraméterlista elemeit ;-vel választjuk el egymástól. (`Procedure P(i: in Integer; c: out Character )`).
Paraméter címzés, vagy mi:S

Deklaráció

"Név pramaéterlista(??) visszatérési érték típusa" a deklaráció általánosan minden nyelven.
Pl.: `int f(int i, bool e)`. Ez ADA-ban a következőképp néz ki:
`Function F(i: integer; b: boolean)Return Integer`.

Szignatúra

Név, paraméterek típusa, és azok sorrendje.

Túlterhelés

Adott név esetén a paraméterek típusának vagy sorrendjének megcserélésével túlterhelhető a függvény. Visszatérési értékre is túlterhelhető

Operátor túlterhelés

```
function "+" (a: Kutya; B: Days) Return Boolean is
...


  := "+"(k,d);
  := k + d;
```

Operátort meghívhatunk függvényként, vagy operátorként is.
`Funcion F(i: integer, b: boolean)Return Integer --..`
`Funcion F(i: integer, p: positive)Return Integer --..`
`F(12, 12)` – kikövetkezteti, hogy a 2. félét kell meghívni. De pl. ha egyiknél `Integer`, másiknál `Positive`, akkor ütközik.

3. Gyakorlat

Tömbök

Nevesített tömbtípusok méretét a típusdefiníciókor kell megadni (Ilyenkor beszélünk definit tömb típusról), deklarációnál nem. Ezeket értékül adhatjuk egymásnak, viszont nevesítetlen tömböknél a definiáláskor adjuk meg a méretüket, de a típusuk nem egyezik meg, így nem lehet értékül adni. Indefinit tömb típusról beszélünk, ha a típusdefinicónál nem adjuk meg a konkrét tömb méretét, csak egy maximum intervallumot. A konkrét érték a deklarációkor kerül megadásra.

```
Procedure..
  T: array(1..10) of Integer; --zárójelben a tömb tartománya.
  T1, T2: array(1..10) of Integer; --Nem lesz egyféle típus, ahhoz létre
     kell hozni egy tömb típust.:
  type Tomb is array(1..10) of Integer;
  T3, T4: Tomb; --Így már egyeznek a típusok. Ezeket lehet értékül adni
     egymásnak.
  type Tomb2 is array(Integer range<>) of Integer;
  T5: Tomb2(1..5);
Begin
  T3 := T4;
End;
```

ADA-ban nincs `[]` operátor a tömbökre.
Több dimenziós tömböket is lehet létrehozni: `type Matrix is arras (1..3, 1..5)of Natural;`

```
Procedure..
  T: array(1..10) of Integer;
Begin;
  for I in T' Range loop -- T'First..T'Last
    --Több dimenziós tömb esetén:
    T'First(1) -- Az első dimenzióbeli elem értéke
```

Ha a tömb 1. elemére akarunk hivatkozni, akkor ezt a `T(T'First)` parancsal tehetjük meg. A `T'First` az intervallum elejét adja meg.

Stringek

A string egy karakterből álló tömb.

```
type String is array(Integer range<>)of Character;
```

Attribútumok

- `name'FIRST(n)`: a tömb n. dimenziójának 1. eleme.
- `name'LAST(n)`: a tömb n. dimenziójának utolsó eleme.
- `name'RANGE(n)`: a tömb n. dimenziójának az index intervalluma.
- `name'LENGTH(n)`: a tömb n. dimenziójának elemeinek száma.

Természetesen 1 dimenziójú tömb esetén a `(n)` elhagyható.

4. Gyakorlat

Record

A record egy inhomogén összetett adatszerkezet. Egész rekordot lehet értékül adni, vagy akár tagonként is.

```
type Complex is record
  Re: Integer;
  Im: Integer;
end record;

c1: Complex;
c1.Re := 5;
c1.Im := 3;
--egyszerre is lehet értéket adni:
c1 := (5,3);
```

Limited record

Egylépésbeli értékadást a rekorda le lehet tíltani: `type Rac is limited record`.
Ekkor `r1 := r2` nem működik, és az egyenlőségvizsgálat sem megengedett.

Diszkriminánsos record

Lehetőség van "paramétert is adni a recordnak. A diszkrimináns csak diszkrét típus lehet, az értékét pedig deklarációkor kell megadni.

```
type name(parameterlist := defaultvalues) Is record
member: type;
...
end record;
```

Variáns record

```
type Status(H; Ö; E; Egy);

type Ember(a: Status) is record
  name: String(1..100);
  Age: Poistive;
  case a is
    when H => Hname: String(..);
    when Ö => H: Integer;
    ...
  end case
end record;

e: Ember(Egy);
e.H := 5; --NEM LEHET!!! hisz nincs neki .H adattagja.
```

5. Gyakorlat

Csomag

Logikailag összetartozó alprogramok, típusdefiníciók, változok stb. halmaza. Az ADA csomagja nem az OOP támogatása, hanem lehetőség a program kisebb, önnáló részekre bontására.
Önálló fordtási egység, amely két részből áll:

1. specifikációs rész (.ads)

2. törzsrész (.adb)

Specifikációs rész

Itt megadjuk a csomagban használni kívánt alprogramokat, típusokat. Az itt definiáltat publikusak lesznek a csomag használója felé.

```
package Name is
--definiciók listája
end Name;
```

Törzsrész

Itt kerülnek implementációra a specifikációs részben deklarált alprogramok.

```
package body Name is
  procedure procedureName1 is
  begin
  --..
  end procedureName;
  --...
end Name;
```

A felhasználás során a with Name; use Name; kulcsszavakat illesztjük be a programba.

Private - Átlátszatlan rész

Célja olyan adatszerkezet létrehozása amit a felhasználó csak használ, a felépítéséről nem tud semmit, tartalmát csak az általunk megadott fügvényekkel éri el.

```
package Name is
--definiciók listája
private
--átlátszatlan rész
end Name;
```

Ha átlátszatlan csomaggal dolgozunk, meg kell adni a típusdefiníciók között: `Type PrivateName is private;`. Ennek a típusnak a kifejtését a csomagunk privát részében kell megadni. Az összes olyan típust, alprogramot, amit nem akarunk publikussá tenni itt helyezhetjük el.

6. Gyakorlat

Sablon

Célja egy olyan általános adatszerkezet leírása, mely lehetőséget biztosít több típus használatára. Sablon lehet eljárás, függvény, csomag is, de nem önálló fordítási egység. Használat előtt példányosítani kell.

```
generic
  --sablonparaméterek
pacakage Name
  --...
end Name;

generic
  --sablonparaméterek
procedure Name(parameters);
```

Sablonparaméter lehet:

- változó,

- típus,

- alprogram.

Típusok esetében meg kell határozni, hogy az egyes típusoknak milyen speciális tulajdonságait szeretnénk kihasználni.

Forrás

- ELTE IK programtervezői informatikus szak 2013 őszi féléves Programozási nyelvek - ADA gyakorlat alapján írt órai jegyzetem.