Ada 95

Snabböversikt av grundläggande språkdelar

Tommy Olsson, Institutionen för datavetenskap, © 1997.

Lexikala element

Separatorer

Radslut, mellanrumstecken och tabulertecken används för att separera lexikala element.

Avgränsare

Avgränsare utgörs av följande specialtecken

```
& ' ( ) * + , - . / : ;  < = > |
```

och följande tvåteckensavgränsare

```
=>  ..  **  :=  /=  >=  <=  <<  >>  <>
```

Identifierare

Identifierar kallas namn på variabler, konstanter, underprogram, paket, processer, datatyper, etc. Första tecknet ska vara en bokstav, för övrigt får *bokstäver*, *siffror* och enkla *understrykningstecken* finnas. Sista tecknet i en identifierare får ej vara ett understrykningstecken.

```
Count  X  Get_line  Ada_95  Make_My_Day
```

Litteraler

Litteraler kallas ”bokstavliga” konstanter. Det finns *numeriska*, *tecken-*, *sträng-* och *uppräknings*literaler.

```
4711  3.14  16#FF#  'A'  "ABC"  True
```

Kommentarer

Kommentarer inleds med dubbla bindestreck och avslutas av radslut.

```
-- Denna kommentar slutar nu!
```

Pragma

Ett pragma är ett direktiv till kompilatorn, t ex hur optimering ska göras.

```
pragma Optimize(Speed);
```

Reserverade ord

```
abort      else       new        return
abs        elsif      not        reverse
abstract   end        null
accept     entry                 select
access     exception              separate
aliased    exit       of         subtype
all                   or
and        for        others     tagged
array      function   out        task
at                                terminate
           generic    package    then
begin      goto       pragma     type
body                  private
           if         procedure  until
case       in         protected  use
constant   is         raise
                      range      when
declare               record     while
delay      limited    rem        with
delta      loop       renames
digits                require
do         mod        requeue    xor
```

Objektdeklarationer och typer

Objektdeklaration

En objektdeklaration inför en *variabel* eller en *konstant* i programmet. Varje objekt har en specifik typ. Konstanter *måste* ges ett värde i deklarationen.

Exempel på objektdeklarationer i ett **`declare`**-block:

```
declare
  I    : Integer;
  PI   : constant Float := 3.1415;
  Mean : Float := (X + Y) / 2.0;
begin
  satser som opererar på objekten (bl.a.)
end;
```

Typer

Typen (datatypen) för ett objekt eller uttryck, avgör vilka värden objektet/uttrycket kan anta och vilka operationer som kan utföras på objektet/värdet.

Följande typer och undertyper är fördefinierade i Adas standardbibliotek:

```
Boolean
Integer    Natural    Positive     Float
Character  Wide_Character
String     Wide_String
Duration
```

Typdeklarationer

Nedan följer ett urval av enklare typdeklarationer.

```
type Level is (Low, Medium, Urgent);
type Short_Integer is range -127..128;
type Hash_Index is mod 997;
type Real is digits 8 range -10.0..10.0;
type Volt is delta 0.01 range -1.0..1.0;

type Vector is
  array(Short_Integer range <>) of Real;

type Buffer(Size : Natural) is
  record
    Position : Natural := 0;
    Value    : String(1..Size);
  end record;

type Vector_Ref is access Vector;
```

Typomvandling

Exempel på typomvandling:

```
Integer(X)   -- X är en variabel av reell typ
```

Uttryck och operatorer

Sammansatta uttrycks beräkningsordning bestäms av operatorernas *prioritet*, *associativitet*, samt *parenteser*. Beräkningsordningen för *operander* definieras i de flesta fall *ej* av språket.

Exempel på uttryck:

```
Sin(X)
not Found
B**2 - 4.0 * A * C
I in Short_Integer
Line(1..3) = "cat"
(Cold and Sunny) or Warm
P /= null and then P.Next /= X
```

Operatorer

```
**   not  abs                    (högsta prioritetsop.)
*    /    rem   mod              multiplikativa
+    -                           enställiga additiva
+    -    &                      tvåställiga additiva
<    <=   =    /=   >=   >      relationsoperatorer
and  or   xor                    logiska (lägst prioritet)
```

Kortslutningsformer

Vänsterledet beräknas först och om detta värde bestämmer hela uttryckets värde beräknas ej högerledet.

```
and then     or else
```

`False` respektive `True` kortsluter uttrycken.

Medlemsskapstest

Testa om ett väde ingår i en viss värdemängd, eller ej.

```
in     not in
```

Attribut

Attribut ger karakteristik om datatyper, objekt, underprogram, m m. (definitionsmässigt är de funktioner).

```
Integer'Last
Integer'Image(I)
Real'Digits
Vector'Range
Date'Size
Sin'Address
```

Grundläggande satser

Tomma satsen

Syntax: **`null`**`;`

Används där en sats krävs men inget ska utföras.

Tilldelningssatsen

Används för att ge en variabel ett (nytt) värde.

Syntax: *variabel* `:=` *uttryck*`;`

Exempel på tilldelningssatser:

```
I := 1;
C := Line(21);
Line(21) := 'A';
Mean := (X + Y) / 2.0;
```

Blocksatsen

Syntax för blocksatsen:

blocknamn`:`
`declare`
`--` *deklarationer*
`begin`
`--` *satser*
`exception`
`--` *undantagshanterare*
`end` *blocknamn*`;`

Exempel på blocksats:

```
…
Get(Key);
begin
  P := new Node;
exception
  when Storage_Error =>
    Put_Line("Minnet slut!");
    Close_Down;
end;
P.Key := Key;
…
```

Selektionssatser

Ada har två grundläggande selektionssatser, **`if`** och **`case`**.

Syntax för **`if`**-satsen (**`else`** och **`elsif`** är valfria):

`if` *booleuttryck* **`then`**
satser
`elsif` *booleuttryck* **`then`**
satser
`elsif` …
…
`else`
satser
`end if``;`

Exempel på **`if`**-satser:

```
if I > Max_Index then
  I := 1;
end if;
```

```
if A > B then
  Max := A;
else
  Max := B;
end if;
```

```
if Key < P.Key then
  Find(Key, P.Left);
elsif Key > P.key then
  Find(Key, P.Right);
else -- vi har hittat Key i noden P
  return P;
end if;
```

Syntax för **`case`**-satsen (**`when others`** är valfri):

`case` *diskret uttryck* **`is`**
`when` *väljarlista* `=>` *satser*
`when` *väljarlista* `=>` *satser*
…
`when others` `=>` *satser*
`end case``;`

väljarlista kan vara uppräkning och/eller intervall, samt även namn på undertyp till väljaruttryckets typ.

Exempel på **`case`**-satser:

```
case Day is
  when Monday..Friday => Work;
  when Saturday       => Party;
  when Sunday         => Rest;
end case;
```

```
case Month is
  when Apr | Jun | Sep | Nov =>
    Number_Of_Days := 30;
  when Feb =>
    if Leapyear then
      Number_Of_Days := 29;
    else
      Number_Of_Days := 28;
    end if;
  when others =>
    Number_Of_Days := 31;
end case;
```

Repetitionssatser

Ada har en grundläggande, sluten slingkonstruktion, **`loop`**-satsen. För att styra slingor finns, dels **`exit`**-satsen

för att avbryta inifrån slinga, dels två s k *iterationsscheman*, **for**- och **while**, med vilka man konstruerar *räknarstyrda* respektive förtestade, *villkorsstyrda* slingor. Eftertestade villkorsstyrda slingor konstrueras med hjälp av **exit**-satsen.

Syntax för **exit**-satsen:

```
exit slingnamn when booleuttryck;
```

Exempel på **exit**-satsen:

```
exit;
exit when I = 0;
exit Main_Loop when All_Done;
```

Syntax för **loop**-satsen med **exit**-sats:

```
slingnamn:
  loop
    satser
    exit slingnamn when booleuttryck;
    satser
  end loop slingnamn;
```

slingnamn är valfritt.

Exempel på **loop**-sats:

```
Input_Choise:
 loop
   Put("Ange menyval (1-5): ");
   Get(Choise);
   exit when 1 <= Choise and Choise <= 5;
   Put_Line("Felaktigt val");
 end loop Input_Choise;
```

Syntax för **for**-iterationschema:

```
slingnamn:
  for styrvariabel in startvärde .. slutvärde loop
    satser
  end loop slingnamn;

for styrvar in reverse start .. slut loop
  satser
end loop;
```

Exempel på **for**-slingor:

```
for I in 1 .. 10 loop
  Put(Vector(I));
end loop;

Print_Backward:
 for I in reverse 1 .. 10 loop
   Put(Vector(I));
 end loop Print_Backward;
```

Syntax för **while**-iterationsschemat:

```
slingnamn:
 while booleuttryck loop
   satser
 end loop slingnamn;
```

Exempel på **while**-slinga:

```
 I := 0;
Input_One_Line:
 while not End_Of_Line loop
   Get(C);
   Line(I) := Upper_Case(C);
   I := I + 1;
 end loop Input_One_Line;
```

Underprogram

Ett underprogram är antingen en *procedur* eller en *funktion*. En funktion returnerar alltid ett värde.

Parametermoder

Det finns tre parametermoder, **in**, **in out** och **out**.

Syntax för parameterdeklarationer:

```
parameternamn : in typ := standardvärde;
parameternamn : in out typ;
parameternamn : out typ;
```

Funktioner kan endast ha parametrar med moden **in**. Denna mod är standardmod och används underförstått om ingen mod uttryckligen deklareras för en parameter.

Syntax för *underprogramdeklarationer*:

```
procedure namn(parameterspecifikation);

function namn(parameterspec.) return värde;
```

Deklarationer av *abstrakta underprogram* avslutas med tillägget "**is abstract**", och i sådana fall finns ingen motsvarande kropp.

Exempel på underprogramdeklarationer:

```
procedure Clear_Screen;
procedure Print(T : in Tree);
procedure Swap(X, Y : in out Float);
procedure Find(X: Key; P: out Position);

function Random return Probability;
function Sin(X : in Real) return Real;
function "*"(Left, Right : in Matrix)
                          return Matrix;
```

Exempel på underprogramanrop:

```
Clear_Screen;
Print(Root);
Swap(A, B);
Find(K, Pos);

while Random < 0.5 loop
  Level := Level + 1;
end loop;

Y := Sin(X);
Matrix_3 := Matrix_2 * Matrix_3;
```

Syntax för *underprogramkroppar*:

```
procedure namn(parameterspecifikation) is
  deklarationer
begin
  satser
exception
  undantagshanterare
end namn;

function namn(parameterspec.) return värde is
  deklarationer
begin
  satser inehållande return värde
exception
  undantagshanterare
end namn;
```

exception-del är valfri, liksom *deklarationer*.

Paket

Alla paket har en *specifikation* men alla har inte en *kropp*.

Syntax för paketdeklarationer:

```
package paketnamn is
   synliga deklatationer
privat
   privata deklarationer
end paketnamn;
```

En **`private`**-del används endast om *privat typ* deklareras i paketet.

Exempel på paketdeklaration:

```
package Key_Pkg is

   type Key is private;

   Null_Key : constant Key;

   function Get_Key return Key;

private
   type Key is new Integer range 0..1000;

   Null_Key : constant Key := 0;
end Key_Pkg;
```

Syntax för paketkroppar:

```
package body paketnamn is
   deklarationer (synliga endast i kroppen)
begin
   paketinitieringssatser (valfritt)
end paketnamn;
```

Exempel på paketkropp:

```
package body Key_Pkg is

   Next_Key := Null_Key;

   function Get_Key return Key is
   begin
      if Next_Key < Key'Last then
         Next_Key := Next_Key + 1;
         return Next_Key;
      else
         return Null_Key;
      end if;
   end Get_Key;
end Key_Pkg;
```

Undantag

Undantag hör inte till grundläggande språkkonstruktioner men kan knappast undvikas, eftersom program kan komma att resa fördefinierade undantag.

Fördefinierade undantag

I paketet `Standard` definieras de fördefinierade undantagen:

```
Constraint_Error     Program_Error
Storage_Error        Tasking_Error
```

Undantag som kan resas av in- och utmatning

I in- och utmatningspaketen finns följande undantag deklarerade:

```
Status_Error         Mode_Error
Name_Error           Use_Error
Device_Error         End_Error
Data_Error           Layout_Error
```

Egendefinierade undantag

Syntax för undantagsdeklaration:

```
undantagsnamn : exception;
```

Exempel på undantagsdeklaration:

```
Key_Error : exception;
```

Resa undantag

Undantag reses med **`raise`**-satsen:

```
raise Key_Error;
raise;    (endast i undantagshanterare för att åter resa)
```

Undantagshantering

Exempel på *undantagshanterare*:

```
…
begin
   Open(File, In_File, "INPUT.DAT");
exception
   when E : Name_Error =>
      Put("Kan inte öppna indatafil: ");
      Put_Line(Exception_Message(E));
      raise;
end;
…
```

```
function Get_Key return Key is
begin
   Next_Key := Next_Key + 1;
   return Next_Key;
exception
   when Constraint_Error =>
      raise Key_Error;
end Get_Key;
```

Fördefinierad in- och utmatning

In- och utmatning i textformat erhålls genom paketet `Ada.Text_IO`. Direkt användbart finns där in-och utmatningsoperationer för tecken och strängar, genom generisk instansiering erhåller man in- och utmatningsoperationer för heltalstyper, modulära heltaltyper, flytpunktstaltyper, fixpunktstaltyper, decimala fixpunktstal typeroch uppräkningstyper. Det finns förinstansierade bibliotekspaket för `Integer` och `Float`, som kan användas genom:

```
with Ada.Integer_Text_IO;
with Ada.Float_Text_IO;
```

Exempel på instansiering av in- och utmatningspaket för olika typer:

```
package Short_Integer_Text_IO is
   new Ada.Text_IO.Integer_IO(Short_Int);

package Boolean_Text_IO is
   new Ada.Text_IO.Enumeration(Boolean);
```

För övrigt finns generiska paket för hantering av (binära) sekventiella- och direktåtkomstfiler, samt paket för strömorienterade operationer på filer.

Exempel på några in- och utmatningsoperationer:

```
Look_Ahead(C, Eol);          -- Character
Get(C);
Put(C);

Get_Line(S);                 -- String
Put_Line(S);

Put(I);                      -- Integer
Put(I, Width => 8);

Put(F);                      -- Float
Put(F, Fore => 6, Aft => 2, Exp => 0);
```