Exempel på ett litet Ada-program

```
-- En kommentar som beskriver något.

with Ada.Text_IO;

procedure Mini is

   -- Deklarationer.
   K    : constant Integer := 5;
   X, Y : Integer;

begin
   -- Körbar kod.
   Ada.Text_IO.Put(”Utskrift”);
   X := 10;
   Y := X * K;  -- Här beräknar vi.
   Y := X + 1;
end Mini;
```

Variabler och konstanter

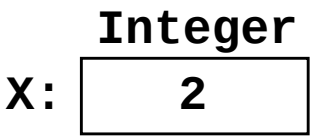
Skall deklareras:

```
X : Integer;
Y : Integer := 2;
Z : constant Integer := 10;
```

Definieras av att de har:

- **ett namn (X)**
- **en datatyp (Integer)**
- **ett värde (t.ex. 2)**

Lagras i datorns minne!



Grundläggande datatyper

Enkla datatyper

`Character`
`Boolean`
`Integer` } **Uppräkningsbara**

`Float`

} **Skalära**

Speciellt i Ada är att man dessutom har ett par extra heltalsdatatyper. Dessa är endast ”`undertyper`” till datatypen `Integer`:

`Natural` **- Noll (0) och uppåt.**
`Positive` **- Ett (1) och uppåt.**

Sammansatta datatyper

`String` **(för att lagra text)**

Enkelt programskelett

```
with ...;
use ...;

procedure ... is

   -- Deklarationer av t.ex. variabler,
   -- konstanter, typer, och underprogram.

begin

   -- Satser som utför något.

end ...;
```

Tilldelning (hård typning)

Generell form på en tilldelningssats:

```
Variabel := Uttryck;
```

Exempel på ett antal tilldelningar i ett program.

```
procedure Tilldelning is

  I : Integer;
  F : Float;
  C : Character;
  B : Boolean;

begin
  I := 7;
  I := 3.14;
  F := 3.14;
  I := Integer(F);
  F := I;
  F := 3 / 2;
  F := Float(3 / 2);
  F := 3.0 / 2.0;
  I := 3 mod 2;
  C := '2';
  I := '2';
  B := True;
  B := (I < 8);
end Tilldelning;
```

Operatorer

Aritmetiska operatorer:

```
+   -   *   /   mod   rem   **
```

Logiska operatorer:

```
and   or   xor   not   and then   or else
in   not in
```

Relationsoperatorer (också logiska operatorer):

```
=   /=   <   >   <=   >=
```

Strängsammanslagningsoperatorn:

```
&
```

OBS! Prioritetsordningen mellan operatorerna kan ändras med hjälp av parenteser precis som i matematiken.

OBS! För relationsoperatorerna krävs parenteser om man blandar olika operatorer (t.ex. `and` och `or`).

OBS! Man kan inte aoutomatiskt utföra de aritmetiska operationerna på data av olika datatyper.

Sekvens

Ett antal satser utförs efter varandra.

```
begin
  Sats_1;
  Sats_2;
  Sats_3;
   ...
  Sats_N;
end;
```

In- och utmatning (eng: I/O) - 1 av 4

För in- och utmatning krävs särskilda paket (OBS! fel in Skansholms bok):

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

procedure In_Och_Utmatning is

  I : Integer;
  F : Float;
  C : Character;
  S : String(1..5);

begin
  Ada.Text_IO.Put(”Mata in: ”);
  Ada.Float_Text_IO.Get(F);
  Ada.Integer_Text_IO.Get(I);
  Ada.Text_IO.Get(C);
  Ada.Text_IO.Get(S);

  Ada.Text_IO.Put(”Utskrift: ”);
  Ada.Float_Text_IO.Put(F);
  Ada.Integer_Text_IO.Put(I);
  Ada.Text_IO.Put(C);
  Ada.Text_IO.Put(S);
end In_Och_Utmatning;
```

In- och utmatning - 2 av 4

Man kan undvika långa rader kod genom att göra ”use” på paketen:

```
with Ada.Text_IO;   use Ada.Text_IO;
with Ada.Float_Text_IO;
use Ada.Float_Text_IO;
with Ada.Integer_Text_IO;
use Ada.Integer_Text_IO;

procedure In_Och_Utmatning_2 is

  I : Integer;
  F : Float;
  C : Character;
  S : String(1..5);

begin
  Put(”Mata in: ”);
  Get(F);
  Get(I);
  Get(C);
  Get(S);

  Put(”Utskrift: ”);
  Put(F);
  Put(I);
  Put(C);
  Put(S);
end In_Och_Utmatning_2;
```

In- och utmatning - 3 av 4

Man kan också ”formatera” utmatningen (inte alla, men vissa) genom att lägga till extra ”parametrar”:

```
procedure In_Och_Utmatning_3 is

  I : Integer;
  F : Float;
  C : Character;
  S : String(1..5);

begin
  Ada.Text_IO.Put_Line(”Mata in: ”);
  Get(F);
  Get(I);
  Get(C);
  Get(S);
  Ada.Text_IO.New_Line(2);

  Put(”Utskrift: ”);
  Put(F, Fore => 3, Aft => 2,
         Exp => 0);
  Put(F, 3, 2, 0);
  Put(I, Width => 6);
  Put(I, 6);
  Put(C);
  Put(S(1..3));
  Ada.Text_IO.New_Line;
end In_Och_Utmatning_3;
```

In- och utmatning - 4 av 4

Några specialare:

```
procedure In_Och_Utmatning_4 is

  S : String(1..5);
  L : Integer;

begin
  Ada.Text_IO.Put_Line(”Mata in: ”);

  Get(S);
  Ada.Text_IO.Skip_Line;
  Ada.Text_IO.Put(S);
  Ada.Text_IO.New_Line;

  Get(S);
  Ada.Text_IO.Skip_Line;
  Ada.Text_IO.Put_Line(S);

  Ada.Text_IO.Get_Line(S, Last =>L);
  Ada.Text_IO.Put_Line(S(1..L));

  Ada.Text_IO.Get_Line(S, L);
  Ada.Text_IO.Put_Line(S(1..L));
end In_Och_Utmatning_4;
```

Val (selektion) - 1 av 2

Man kan välja att utföra en av en eller flera sekvenser av satser beroende på olika villkor eller att inte utföra någon.

```
if Villkor then
  Sekvens_Av_Satser;
end if;
```

```
if Villkor then
  Sekvens_Av_Satser;
else
  Sekvens_Av_Satser;
end if;
```

```
if Villkor then
  Sekvens_Av_Satser;
elsif villkor then
  Sekvens_Av_Satser;
end if;
```

```
if Villkor then
  Sekvens_Av_Satser;
elsif villkor then
  Sekvens_Av_Satser;
else
  Sekvens_Av_Satser;
end if;
```

Val (selektion) - 2 av 2

Man kan välja att utföra en av en eller flera sekvenser av satser beroende på ett uttryck.

```
case Uttryck is
  when ... => Sekvens_Av_Satser;
   ...
  when ... => Sekvens_Av_Satser;
end case;
```

```
case Uttryck is
  when ... => Sekvens_Av_Satser;
   ...
  when others => Sekvens_Av_Satser;
end case;
```

Repetition (iteration)

Att upprepa något ett antal gånger. Tre olika sätt att göra detta i Ada.

```
for Styrvariabel in [reverse]
    Min_Värde .. Max_Värde loop
  Sekvens_Av_Satser;
end loop;
```

```
while Villkor loop
  Sekvens_Av_Satser;
end loop;
```

```
loop
  Sekvens_Av_Satser;
end loop;
```

I alla dessa varianter kan man utföra en speciell sats för att avbryta repetitionen (`exit`) om man vill. Denna kan utformas på lite olika sätt.

Sekvens (med lokala deklarationer)

Ett antal satser utförs efter varandra där man kan ha lokala variabler, typer m.m. som alltså bara existerar i denna sats.

```
declare
  -- Lokala deklarationer.
begin
  Sekvens_Av_Satser;
end;
```

Enkla underprogramskelett

En procedur:

```
with ...;
use ...;
procedure ...(...) is
   -- Deklarationer av t.ex. variabler,
   -- konstanter, typer, och underprogram.
begin
   -- Satser som utför något.
end ...;
```

En funktion:

```
with ...;
use ...;
function ...(...)
          return ... is
   -- Deklarationer av t.ex. variabler,
   -- konstanter, typer, och underprogram.
begin
   -- Satser som utför något.
   return ...;
end ...;
```