Hoofdstuk 0: Inleiding | Pagina 1-3

Pascal Borland
- *Klassiek:* declaraties, procedures en functies
- *Abstracte datatypes:* onderscheid tussen publieke interface en private implementatie
- *Objectgeoriënteerde paradigma:* object, overerving, boodschappen, ...

Units: hergebruik van routines (user interfaces mogelijk)
Event-Driven Programming: niet sequentieel, wel reagerend op gebeurtenissen

Delphi: componentgebasseerde ontwikkelingsomgeving
--> oplossing voor user interface

Hoofdstuk 1: Kennismaking met programmeertalen | Pagina 5-9

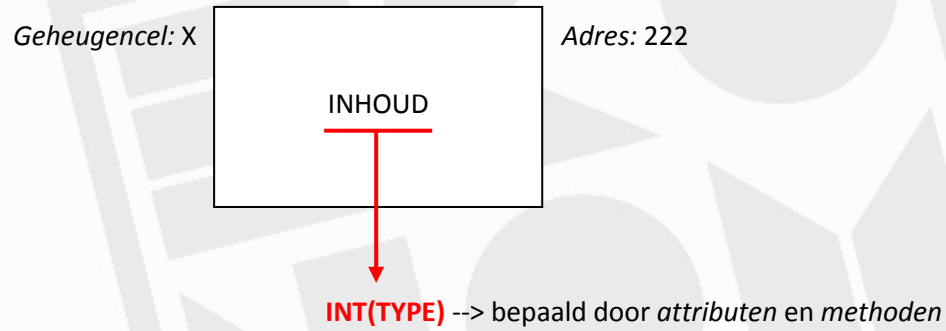
Voorbeeld doorheen hoofdstuk:
Teller: verhogen, verlagen, herzetten
Verschillende talen: Java, Eiffel en Pascal

Objectgeoriënteerde programmeertalen
--> Object: logische entiteit in computergeheugen

Object bevat **attributen**
--> locatie voor opslag van gegevens

Gegevens worden bepaald door **methodes**
--> manipulatie van gegevens

Klasse: beschrijving v/h type v/h object



Operaties: wat men kan doen met het type (verhogen, lagen, herzetten, ...)

www.ekonomika.be Dekenstraat 2, 3000 Leuven - info@ekonomika.be

TELLER: ONTLEDING IN PASCAL

KLASSE TELLER

```
unit UTeller;                          -- Declaratie groep (niet belangrijk)
interface                              -- Duiden dat het om een interface gaat (n.b.)

type                                   -- Declaratie module functies, procedures en class
  TTeller = class(TObject)             -- Definiëren van de klasse
      private (--> onzichtbare gegevens, declaraties)
          Stand: integer;              -- Declaratie v/h type v/d inhoud (geheel getal)
      public (--> opsomming zaken in implementatie)
          constructor Create;          -- Aanmaak procedure vooraf "constructor"
          procedure Verhogen;          -- Procedure (operatie)
          procedure Verlagen;          -- Procedure (operatie)
          procedure Herzetten;         -- Procedure (operatie)
          function GeefStand: integer; -- Procedure met 'result' is "function"
          procedure StandOutput;       -- Procedure (operatie)
  end;                                 -- Afsluiten v/d klasse TTeller

implementation                         -- Implemantie van zaken uit public

constructor TTeller.Create;            -- Aanmaken teller (constructor klasse.Create)
  begin                                -- Elke implementatie start met "begin"
      inherited Create;                -- Create procedure standaard
      Stand := 0;                      -- Declaratie stand op 0 (gelijkstellen)
  end;                                 -- Elke "begin" staat in combinatie met "end"

procedure TTeller.Verhogen;            -- procedure + klasse.actie
  begin
      Stand := Stand + 1;              -- Operatie met manipulatie
  end;                                 -- Einde v/d methode

procedure TTeller.Verlagen;            -- procedure + klasse.actie
  begin                                * Condities tussen haakjes weergeven
    if (stand > 0) then Stand := Stand -1; -- Controle gevolgd door manipulatie*
  end;                                 -- Einde v/d methode

procedure TTeller.Herzetten;           -- procedure + klasse.actie
  begin
      Stand := 0;                      -- Operatie met manipulatie
  end;                                 -- Einde v/d methode
```


www.ekonomika.be Dekenstraat 2, 3000 Leuven - info@ekonomika.be

```
function TTeller.GeefStand: integer;        -- function + klasse.actie: type
  begin
        Result := Stand;                    -- Result zorgt dat het om functie gaat
  end;                                      -- Einde v/d methode

procedure TTeller.StandOutput;              -- procedure + klasse.actie
  begin
        writeln('De stand van de teller is', Stand)  -- Afdrukken v/d stand *
  end;                                      -- Einde v/d methode
end.                                        -- Einde v/d implementatie
```

* Om af te drukken gebruiken we als standaard
writeln(...**)** met tussen de haakjes wat afgedrukt moet worden
- *vb. Stand:* dit drukt de waarde af die gegeven is aan stand (en niet het woord)
- *vb. '...':* dit drukt de letterlijke boodschap af tussen de haakjes

Stel: Stand is gelijk aan 5
writeln('Stand**')** resulteert in het woord Stand
writeln(Stand**)** resulteert in het getal 5 (de waarde van stand)

MANIPULEREND PROGRAMMA

```
program PTeller;                            -- Declaratie dat het om uitvoeren gaat
uses
  Uteller in 'Uteller.pas';                 -- Niet belangrijk

var                                         -- Declaratie van de variabelen
  teller: TTeller,;                         -- Woord teller gelijk stellen met klasse
  i: integer;                               -- Aantal keer tellen als geheel getal

begin                                       -- Eigenlijke uitvoer van het programma
teller := TTeller.Create;                   -- Teller aanmaken (verwijst create klasse)
for i := 1 to 3 do teller.Verhogen;         -- Tellerstand drie maal verhogen* (LOOP)
        teller.StandOutput;                 -- Tellerstand tonen*
        teller.Verlagen;                      .
        teller.StandOutput;                   .
        teller.Herzetten;                     .
        teller.StandOutput;
        teller.Verlagen;
        teller.StandOutput;
        teller.free;
        readln;                             -- Lezen
end.                                        -- Programma stoppen
```

* Merk op dat men enkel bij .create nog de klassenaam (TTeller) gebruikt

www.ekonomika.be Dekenstraat 2, 3000 Leuven - info@ekonomika.be

Hoofdstuk 2: Begrippen in de OOP-Wereld | Pagina 16-32

Gestructureerd en objectgeoriënteerd programmeren zijn complementair

Vb: Selectie - exclusief
- exhaustief

OOP:
- Objecten kennen
- Methodes kennen
- Communicatie kennen

Object: natuurgetrouwe representatie van een al dan niet tastbaar begrip uit de werkelijkheid (*kortom:* alles, ook zaken als liefde, toekomst)

Definitie: Object

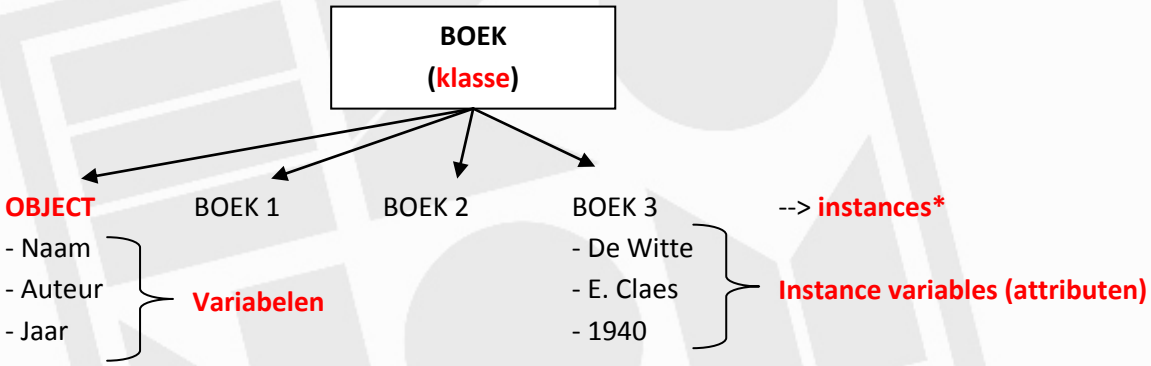
Object is een geïntentieerde entiteit bestaand uit
- Set operaties (object gehoorzaamt)
- Toestanden (beïnvloed door operaties)
- Mogelijkheid tot versturen boodschappen (naar ander object en operateur)

Voorbeeld:
Object: boek
Attributen (**instance variables**): naam, jaar, kernwoord, auteur, uitgeverij

Sommige zaken zijn op zich terug opbjecten: auteur, uitgeverij, ...

Voorbeeld:
Auteur is een instance variabele v/e boek
--> zelf ook object met instance variabelen (adres, mail, leeftijd, ...)

Klasse: generalisatie van alle objecten met gelijksoortige eigenschappen



* Instances zijn objecten met feitelijke verschijningsvormen in een klasse
Klasse: abstract, niet concreet **Object:** tastbaar

www.ekonomika.be Dekenstraat 2, 3000 Leuven - info@ekonomika.be

Definitie: Klasse

Klasse: template die eigenschappen v/e object specificieert
--> eigen interface met omschrijving algemene eigenschappen (toegankelijkheid ↑)

Class body: Implementeert operaties (omschreven in de interface)
Instance: variabelen die de toestand implementeert
Instance variables: attributen v/e object
--> geen rechtstreekse benadering, nood aan boodschappen

Definitie: Message

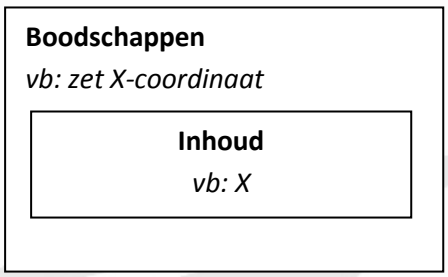
Message is een boodschap naar het object die zal leiden tot een bepaalde actie v/h object

Message
- Communicatie tussen object en programmeur
- Communicatie tussen objecten (ontvangst boodschap: handelingen)

Information hiding
Afschermen informatie v/e object en implementatiedetails v/d buitenwereld
--> messages zijn externe representatie v/h object (en bescherming interne info)

Inhoud v/e object (attributen) zetten **encapsulation** rond de methoden die messages aanroepen (attributen zijn nl. slechts mogelijk via methodes v/h object)

Boodschappen laten *wel* toe om objecten aan te spreken



Soort messages
- Opdracht gevende messages: veranderen de inhoud v/h object
vb: zet X-coordinaat 15

- Informatie opvragende messages: willen inhoud tonen
vb: geef X-coordinaat

Wanneer een object een opdracht krijgt
--> men gaat iets moeten doen
Bepaald door **methode:** stuk code die uitvoering bepaald

www.ekonomika.be Dekenstraat 2, 3000 Leuven - info@ekonomika.be

Definitie: Methode

Methode is verantwoordelijk voor eigenlijke uitvoering v/d actie
(*reactie op de boodschap of message*)

Object heeft zo:
- Inhoud: attributen
- Functionaliteit: methodes die attitudes manipuleren

Vb: zet X-coordinaat 15 zal een methode lanceren
- Methode start
- Locatie v/d pen wordt bepaald
- Controle v/h argument (15)
- ...

Eén message kan verschillende methodes oproepen
Een methode hoort wel uniek bij een message

Vb: stel er zijn twee pennen, de actie *zet X-coordinaat* zal voor beide pennen iets doen

Inheritance of **overerving:** klasse is generalisatie of specialisatie v/e andere klasse

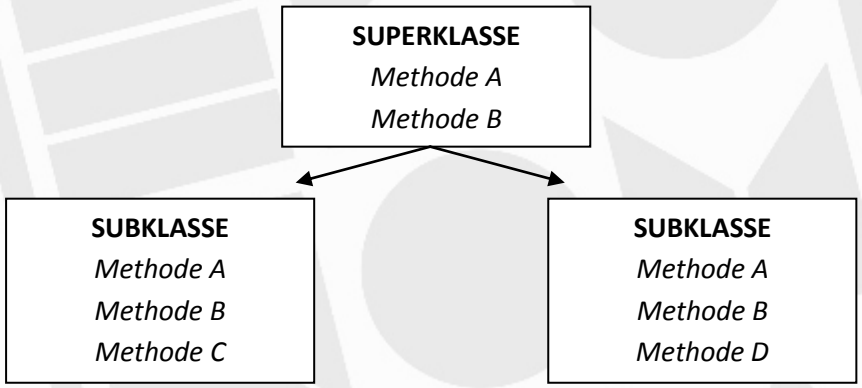
Vb:
BELG is **superklasse/ouderklasse/generalisatie**
van GENTENAAR dat op zijn beurt een **subklasse/kinderklasse/specialisatie** is

Extra voorbeeld op pagina 23

Definitie: Inheritance

Inheritance impliceert dat de verschijningsvorm v/e klasse, een intantie, alle eigenschappen heeft van de klasse plus alle eigenschappen v/e eventuele superklasse

Een subklasse zijn object erft v/e hogere klasse:
- De inhoud
- De messages



www.ekonomika.be Dekenstraat 2, 3000 Leuven - info@ekonomika.be

Er zijn twee vormen van overerving

- Strikte overerving
- Niet-strikte overerving

Strikte overerving

Klasse erft van andere klasse

De subklasse is meer gespecialiseerd dan de superklasse, maar bevat alles v/d superklasse

Voorbeeld:

SPEELGOED	**PUZZEL**
+ Prijs	**+ Prijs**
+ Leeftijd	**+ Leeftijd**
+ # Spelers	**+ # Spelers**
	+ # Stukjes
	+ # Duur

Definitie: Strikte overerving

Strikte overerving doet zich voor als de subklasse alle eigenschappen v/d superklasse overneemt (mogelijk met andere implementatie) en zelf nog bijkomende kenmerken ter specialisatie toevoegt

Niet-strikte overerving

Definitie: Niet-strikte overerving

Niet-strikte overerving komt voor als bepaalde kenmerken v/d superklasse niet voorkomen of anders genoemd zijn in de subklasse

Er zijn drie verschillende opties:

- **Gewone weglating** of **hernoeming:** eigenschappen v. één superklasse, mogelijk hernoemt

Voorbeeld: hernoeming Num_Players -> Ideal_Num_players

Problemen: potentieel tot run-time error

- **Meervoudige overerving:** subklasse erft van twee of meerdere superklassen

--> zal leiden tot een optelsom in de subklasse v/d twee bovenliggende klassen

Problemen:
- Identieke kenmerken uit beide superklassen zijn mogelijk (incorrect)
- Geen eenvoudige hiërarchische structuur meer
- Verandering in superklasse zal leiden tot problematiek
- **Niet elke OO-taal laat meervoudige overerving toe**

- **Herhaalde overerving:** subklasse stemt meer dan één keer v/d superklasse af

BEKIJK MODEL PAGINA 25

www.ekonomika.be Dekenstraat 2, 3000 Leuven - info@ekonomika.be

Definitie: Polyformisme

Polyformisme duidt op het feit dat dezelfde messages naar objecten van geheel verschillende klasse gestuurd kunnen worden

Vb: zet_kleur_rood kan zowel voor een pen als een auto
--> de afhandeling zal wel anders zijn

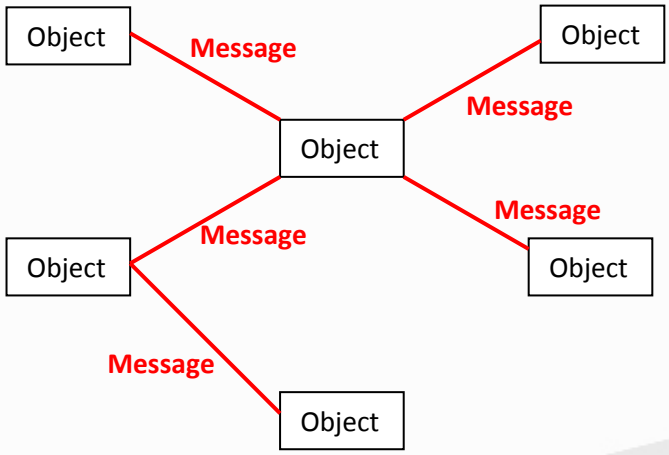
Er zijn twee soorten polyformisme:

- Object polyformisme
- Kenmerk polyformisme

Verschil zit zich in het *oproepen* en *opslagen* v/d objecten

OPEN BOEK 28 - 29

Voorbeeld: OO-Programma



OVERZICHT DEFINITIES TE VINDEN OP 30 - 31

www.ekonomika.be Dekenstraat 2, 3000 Leuven - info@ekonomika.be

Hoofdstuk 3: Datastructuren en algoritmen in Pascal | **Pagina 32 - 97**

-- Alle tekst in het **blauw** is codering --

3.1 Syntax: Een eenvoudig programma in Pascal | Pagina 32 - 35

-- Bekijk programma voorbeeld pagina 32 - 34 --

Elk programma start met **program + naam programma;**

De statements staan tusen **begin** en **end.** | **!** Let op: punt na laatste end

De statements *scheiden* we door **;**

Er zijn **verschillende statements:**

- Toewijzing
- Selectie
- Iteratie
- Methode-oproep

! Als een methode argumenten heeft, staan deze tussen haakjes

vb: **TRekening.verhoog(**bedrag: **integer)**;

! Binnen één methode onderscheiden we meerdere argumenten via **,**

vb: **TRekening.create(**RNummer_in**,** RStand_in**)**

(beiden argumenten zijn nodig om een object v/d klasse te maken)

Het woord **var** duidt op de *start v/d declaratie van variabelen*

Var gaat dus gevolgd door **naam_variabele: type;**

vb: var nieuwe_naam: string;

Er zijn verschillende **types**, allen geschreven in *kleine letters*:

- integer
- string
- char
- boolean
- real
- eigen_type | doen we via het woord **type**

vb: **type TRekening = class**

Men kan ook **commentaar** toevoegen via:

{commentaar komt hier}

//commentaar komt hier

Inlezen doet men via **read(**variabele**)**

Schrijven doet men via **write(**tekst of variabele**)**

www.ekonomika.be Dekenstraat 2, 3000 Leuven - info@ekonomika.be

3.2: Syntax: Structuur van een programma | Pagina 35

Pascal-programma bestaat uit:

- Klassedefinities
- Methoden
- Hoofdprograma

Algemeen skelet v/e programma

```
program naam;

type    Mijnklasse = class(superklasse)
        attribuut: type;
        procedure Mijnmethode
        end;

procedure Mijnklasse.Mijnmethode;
declaraties;
  begin
        statement;
        statement;
  end;

  begin                  //hoofdprogramma
        statement;
        statement;
  end.
```

Algemeen voorbeeld van klassedefinitie

```
type    TREKENING = class(superklasse)                    -- Declaratie v/d klasse
        RNummer: integer;                                 -- Declaratie v/d variabelen
        RStand: integer;
        constructor create(RNummer_in, RStand_in: integer)  -- Declaratie aanmaak
        procedure verhoog(bedrag: integer);               -- Declaratie v/e methode
        procedure verlaag(bedrag: integer);               -- Declaratie v/e methode
end;
```

Algemeen voorbeeld v/d uitwerking v/e procedure

```
procedure TRekening.verhoog(bedrag: integer);
begin
        RStand := RStand + bedrag
end;
```


www.ekonomika.be Dekenstraat 2, 3000 Leuven - info@ekonomika.be

3.3: Syntax: Uitwerking van een methode | Pagina 36

Methode: mini-programma in een programma
! Kan opnieuw declaraties bevatten
Blok: declaraties, begin, statements, end

Statements
Er zijn verschillende soorten statemenets

Toewijzing
vb: **RStand := RStand + bedrag**

Selectie
```
vb: If KlantenLijst.Aantalklanten = 0                          -- Voorwaarde
        then WriteIn('Er bevinden zich geen klanten in het systeem')   -- Uitvoering
    else                                                       -- Alternatief
        begin                                                  -- Meerdere zaken
        ...
        end
```
! Na THEN en ELSE komt er slechts één statement
--> Meerdere zijn ook mogelijk via: **begin ... end**

Iteratie
```
vb: for i := 1 to Klantenlijst.Aantalklanten do        -- Doorlopen gebied
        WriteIn(F, KlantenLijst.Elementen[i].naam)
```
! Na DO komt er slechts één statement
--> Meerdere zijn ook mogelijk via: **begin ... end**

Oproep (Procedure of Functie)
vb: Klant.Rekening.verhoog(bedrag);
! Een functie is een procedure die een resultaat teruggeeft v/e bepaald type
(**zie voorbeeld teller**)

3.4 Syntax: Meer uitgebreide beschrijving van de Pascal syntaxis | Pagina 36 - 42

BEKIJK PAGINA 36 - 42
(basisprincipes)

www.ekonomika.be Dekenstraat 2, 3000 Leuven - info@ekonomika.be

3.5 Datatypes

Variabele heeft *naam* die *refereteert* naar *waarde* in geheugen
Het type kan een verzameling van waarden en operaties zijn

Er zijn drie soorten datatypes

- **Enkelvoudige data types**
- **Pointers**
- **Samengestelde data types**

Enkelvoudige data types

Dit zijn **scalar types** die maar één waarde kunnen aannemen
Er zijn twee grote groepen

- **Ordinale types:** alle scalar types uitgezonder *real*
- **Data type real**

De **ordinale types** worden verder onderverdeeld in:

- **Voorgedefinieerde types**: integer, boolean en char
- **Enumeratie types**
- **Deelinterval types**

ZIE PAGINA 43 OVERZICHT

Een andere indeling is ook mogelijk:

- **Arithemic types**: integer, real (getallen)
- **logical types**: boolean
- **Hardware-supported types**

ORDINALE TYPES

Types die een **volgorde** bevatten
Doel: vergelijk van twee waarden, opvolger of voorganger te duiden
! Waarden zijn gegroepeerd in een eindige, geordende verzameling

Basisbewerkingen:

succ(x) : Opvolger van x
pred(x) : Voorganger van x
ord(x) : Rangnummer van x
Alle **relationele bewerkingen** zoals **=, <>, <, <=, >, >=**

www.ekonomika.be Dekenstraat 2, 3000 Leuven - info@ekonomika.be

ORDINALE TYPES: Voorgedefinieerde types

INTEGER: Geheel getal, we kunnen hier ordinale operaties op uitvoeren

vb: Succ(5) is 6

Ook **rekenkundige operaties** zijn mogelijk

! Problemen met deling, **/** als deling kan zorgen voor getal met rest (is reëel, geen INTEGER)

Overzicht operaties:

x **+** y

x **-** y

x ***** y

- x

x **div** y | geeft het resultaat v/e deling zonder rest (bv: 8 div 3 = 2) want 3 raakt 2 keer in 8

x **mod** y | geeft de rest na de deling door y (y > 0) (bv: 8 mod 3 = 2) want 3 x 2 = 6 en 8 - 6 = 2

succ(x**)** = x + 1

pred(x**)** = x - 1

ord(x**)** = x

-maxint(...) geeft het minimum en **maxint**(...) geeft het maximum

BOOLEAN: Een variabele van dit type kan enkel **true** (1) of **false** (0) aannemen

Overzicht operaties:

not x

x **and** y

x **or** y

x **≤** y | implicatie (x => y) als x dan y

x **=** y | equivalentie (x = y) y als en slechts als x

x <> y | exclusiviteit (OF)

X	Y	NOT X	X and Y	X or Y	X ≤ Y	X = Y	X <> Y
FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	TRUE	FALSE
FALSE	TRUE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE
TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE
TRUE	TRUE	FALSE	TRUE	TRUE	TRUE	TRUE	FALSE

www.ekonomika.be Dekenstraat 2, 3000 Leuven - info@ekonomika.be

CHAR: Elk karakter dat mogelijk is (letters en cijfers)

Overzicht operaties:

ord(x) | rangnummer v/h tekenset
chr(n) | teken met rangnummer n (*vb:* chr(54) geeft als resultaat a)
succ(x) | volgend teken uit de tekenset
pred(x) | vorig teken uit de tekenset
alle relationele operaties

Char kan bestaan uit:

- **cijfers** (0-9)
- **hoofdletters** (A-Z) | Moeten niet noodzakelijk opvolgen
- **kleine letters** (a-z)

! Cijfers en letters kan men niet vergelijken

ORDINALE TYPES: Enumeratie types

Men bepaald zelf alle waarden mogelijk binnen een type
--> elke waarde in de lijst komt overeen met een orde
vb: Weekdagen = (Maandag, Dinsdag, Woensdag, Donderdag, Vrijdag, ...)
Zal als **ord(**maandag**)** resulteren in **0** en **ord(**dinsdag**)** resulteren in 1

De verzameling is dus **geordend** in de volgorde opgegeven
(naam1, naam2, ..., $naam_n$)

Operaties als **ord**, **succ** en **pred** zijn mogelijk
vb: **succ**(maandag) = dinsdag

Ook *relationele operaties* kunnen, zo zal: maandag < dinsdag TRUE opleveren
! Maandag = lente mag niet, zijn twee verschillende types)

write(maandag**)** kan of mag niet, want maandag is GEEN string

www.ekonomika.be Dekenstraat 2, 3000 Leuven - info@ekonomika.be

Voorbeeld programma

```
program Weekcijfers (input, output);

const   Hoogste = 100;
        Laagste = -Hoogste;

Type Dagen = (Maandag, Dinsdag, Woensdag, Donderdag,
              Vrijdag, Zaterdag, Zondag);          -- Declaratie alle dagen
     Werkdagen = Maandag..Vrijdag;                 -- Declaratie subklasse

Var     Verkopen: ARRAY[Werkdagen]
        Minimum, Maximum: Laagste..Hoogste;
        Dag: Werkdagen

Begin
        For Dag := Maandag To Vrijdag Do
            read(Verkopen[dag]);

        Minimum := Verkopen[Maandag]              -- Initieel maximum
        Maximum := Minimum;                       -- Initieel minimum

        For Dag := Dinsdag To Vrijdag Do          -- Iteratie starten
            If Verkopen[Dag] < Minimum            -- Testen conditie
              Then Minimim := Verkopen [Dag]      -- Gelijk stellen na succes
              Else If Verkopen[Dag] > Maximum     -- Testen conditie
                   Then Maximum := Verkopen[Dag]; -- Gelijk stellen na succes

        WriteIn ('Minimum = ',Minimum);           -- Minimum printen
        WriteIn ('Maximum = ',Maximum);           -- Maximum printen
End.
```

Zoals eerder vermeld:
Werkdagen is een **subrange** (deelverzameling) van dagen

ORDINALE TYPES: Subrange types

Deelintervallen (zoals hierboven) bevorderen enkel de leesbaarheid
Daarnaast bespaart het geheugen

www.ekonomika.be Dekenstraat 2, 3000 Leuven - info@ekonomika.be

REAL

Het gaat hier om reële getallen

Overzicht operaties:
ord(x**)**, **pred(**x**)** en **succ(**x**)** zijn niet meer mogelijk

Alle *rekenkundige operaties* **+**, **-**, *****, **/** zijn wel mogelijk

Alle *relationele operaties* blijven ook mogelijk

! 1. schrijft men als 1.0 in Pascal

! .1 schrijft men als 0.1 in Pascal

--> beiden zullen anders een syntax error opleveren

Samengestelde types

Er is een opsplitsing in twee grote groepen:

- **Vast aantal elementen**
  - Homogeen: ARRAY
  - Heterogeen: RECORD
- **Variabel aantal elementen**
  - Homogeen: SET en FILE

ZIE VOORBEELD BEGIN UITBREIDING (50 - 54)

(meerdere klanten zijn nu mogelijk, maximaal 10, staan in Array)

ARRAY TYPE

Array: lijst van vast aantal elementen v/h zelfde type (vast - homogeen)

Voorstelling: **Array[**IndexType**] of** ComponentType

--> *IndexType* wordt gebruikt om element uit lijst te halen

! Moet een ordinal type zijn (IndexType bepaald # elementen in Array)

--> *ComponentType* duidt op welk type de elementen zijn (kan alles zijn)

Voorbeeld: **Array[**1..MaxKlanten] **of** TKlant;

Een **array** kan van een **hogere dimensie** zijn

--> bepaald door het aantal IndexTypes

Voorbeelden: **Array[...] of Array[...] of Integer**

We kunnen dit gemakkelijker herschrijven als een **matrix**

Array[... , ...] of Integer

Array: afbeelding v/d waarde v/h IndexType op waarden v/h componenttype

Overzicht operaties

- Selecteren (en wijzigen) v/e element

Men kan **selecteren** via bijvoorbeeld **A[i]**, **A[i+j]**, **A[i,k]** (of A[i][k])

- Toekenning v/e ganse Array (kopieëren) of van elk element afzonderlijk

www.ekonomika.be Dekenstraat 2, 3000 Leuven - info@ekonomika.be

RECORD TYPE

Record: vast aantal componenten van mogelijk verschillende types (vast - heterogeen)
! Elk veld of element heeft zijn **eigen type**

Overzicht operaties:
- Selectie van component (veld)
- Assignment

Voorbeeld: Eenvoudig model

```
Type datum =
        Record                          -- Jaren ≠ maanden ≠ dagen
          Jaar: 1900..2100;             -- Vast aantal jaren
          Maand: (Jan, Febr, ..., Dec); -- Vast aantal maanden
          Dag: 1..31;                   -- Vast aantal dagen
        End;
Var Begindatum: Datum;
...
BeginDatum.Jaar := 1987; BeginDatum.Maand := Febr; BeginDatum.Dag := 7
EindDatum := BeginDatum;
Write (EindDatum.Jaar)
```

RECORD TYPE: Records binnen andere structuren

Voorbeeld: Geneste Records

```
Type datum =
        Record                          -- Jaren ≠ maanden ≠ dagen
          Jaar: 1900..2100;             -- Vast aantal jaren
          Maand: (Jan, Febr, ..., Dec); -- Vast aantal maanden
          Dag: 1..31;                   -- Vast aantal dagen
        End;

    persoon =
        Record                          -- Naam ≠ datum
          Voornaam, Naam: ...;
          GeboorteDatum: Datum          -- Maakt gebruik van datum =
        End;

Var Pieter, Pol: Persoon;
Piet.GeboorteDatum.Jaar := 1960;
Pol.GeboorteDatum := Piet.GeboorteDatum
```


www.ekonomika.be Dekenstraat 2, 3000 Leuven - info@ekonomika.be

Voorbeeld: Arrays of Records

```
Var Deelnemer: ARRAY[1..Max] Of Persoon;
...
Deelnemer[i].Naam := ... ;
Deelnemer[j].GeboorteDatum.Jaar := ...
```

RECORD TYPE: Records met variabele componenten

! Afhankelijk v/d waarde v/e bepaald veld, bevat een record een variant gedeelte

--> bestaat mogelijk uit verschillend aantal velden van mogelijk verschillende types

! Slechts één variant gedeelte is mogelijk (op het einde, kan wel genest zijn)

```
Type datum =
        Record                          -- Jaren ≠ maanden ≠ dagen
          Jaar: 1900..2100;             -- Vast aantal jaren
          Maand: (Jan, Febr, ..., Dec); -- Vast aantal maanden
          Dag: 1..31;                   -- Vast aantal dagen
        End;

    persoon =
        Record                          -- Naam ≠ datum
          Voornaam, Naam: ...;
          GeboorteDatum: Datum          -- Maakt gebruik van datum =
          Geslag: (Man , Vrouw);
            Case BurgerlijkeStand: Status of
                Gehuwd, Weduwnaar: (HuwDatum : Datum);
                Gescheiden : (SDatum : Datum);
                Alleenstaand : ()
        End;
...
If Piet.BurgerlijkeStand = Gehuwd
Then Piet.HuwDatum :=
```


www.ekonomika.be Dekenstraat 2, 3000 Leuven - info@ekonomika.be

RECORD TYPE: Records met Statement

Dit statement maakt het mogelijk niet altijd te moeten verwijzen naar record variabele

WithStatement = **With** variablelist **Do** statement

Voorbeeld: Uitgebreid

```
begin
        WriteIn('Klant: ', KlantenLijst.Element[i].Naam);
        WriteIn('RNummer: ', KlantenLijst.Element[i].Rekening.RNummer);
        WriteIn('RStand van de rekening: ', KlantenLijst.Element[i].Rekening. RStand);
end;
```

Voorbeeld: ***Via Statement***

```
begin
        WriteIn('Klant: ', Naam);
        WriteIn('RNummer: ', Rekening.RNummer);
        WriteIn('RStand van de rekening: ', Rekening.RStand);
end;
```

Zo kunnen we ook combinaties herschrijven:

Piet.BeginDatum.Jaar := 1988

wordt

```
With Piet Do
        With BeginDatum Do
                Jaar := 1988
```

wordt

```
With Piet, BeginDatum Do
        Jaar := 1988
```

SET TYPE

Variabel aantal elementen, maar van hetzelfde type (variabel - homogeen)

Verzameling: groep elementen die onderling verschillend zijn met een irrelevante volgorde

Weergave: **Set Of** BaseType

! BaseType moet een **ordinaal type** zijn en het aantal elementen moet kleiner of gelijk zijn aan het aantal in het BaseType.

Voorbeelden:

```
Cijfers = Set Of '0'..'9'
Hoofdletters = Set Of 'A'..'Z'
LangeMaanden = Set Of Maanden          -- ! aantal LangeMaanden < aantal Maanden
```


www.ekonomika.be Dekenstraat 2, 3000 Leuven - info@ekonomika.be

Waarden v/h **Set Type** (de verzameling) duiden we via **opsomming** als:
[Opsomming elementen]
Voorbeelden:
[april, juni, september, november]
[1..10]
['a'..'z','A'..'Z']
[ ] is een **lege set**

SET TYPE: Constructie

! ['a'..'z'] is gelijk aan ['a', 'b', 'c', ..., 'z'] en is gelijk aan ['z', 'y', ..., 'b', 'a']
--> **volgorde heeft geen belang**
! ['z'..'a'] is een **lege set**
--> Het laagste element in deze set, 'z', is nl. van hogere orde dan het hoogste 'a'
! [1, 2, 3, 1] = [1, 2 ,3] want het element 1 komt meermaals voor, mag niet, geschrapt

SET TYPE: Relationele operaties

Stel: A en B, set expressies van hetzelfde type X

Gelijkheid	A = B
Ongelijkheid	A <> B
Deelverzameling van	A <= B
Omvatten	A >= B
Element van	x in A

Voorbeelden:
[Rood, Blauw] = [Blauw, Rood]
['a'..'z'] <> ['A'..'Z']
[1, 2, 3] <= [1..3]
[x] >= []
y in [x, y, z]

www.ekonomika.be Dekenstraat 2, 3000 Leuven - info@ekonomika.be

FILE TYPE

File Type vormt een sequentie van een willekeurig aantal componenten van hetzelfde type
--> **File Of** Type

De declaratie v/e *file f* van type T impliceert
- Een buffervariabele f^ van type T (venster op bestand) | ?
- Een bestandwijzer (plaats in bestand)

Overzicht operaties:

Reset(f**)**	Open f voor lezen v/d eerste component
Rewrite(f**)**	Open f voor overschrijven v/d eerste component
Get(f**)**	Gaat naar de volgende component en plaatst eventuele waarde in buffervariabele
Put(f**)**	Voegt de inhoud v/d buffervariabele toe aan het bestand (achteraan)
Eof(f**)**	Functie die aanduidt of de bestandswijzer zich bevindt na de laatste component

FILE TYPE: Gecombineerde file operaties (Read, Write)

Var f: **File Of**

Voorbeeld: Verbergen v/d buffervariabele

```
Type Metingen = File Of Real;

Var DataIn, DataOut: Metingen;
    Waarde: Real;

Begin
    Reset (DataIn);
    Rewrite (DataOut);
    While Not Eof (DataIn) Do
        Begin
          Read (DataIn, Waarde);          -- Inlezen
          Verwerk (Waarde);
          ...;
          Write (DataOut, Waarde)         -- Kopiëren
        End;
    ...;
    Bepaal (Waarde);
    Write (DataOut, Waarde);              -- Toevoegen
    ...
End.
```


www.ekonomika.be Dekenstraat 2, 3000 Leuven - info@ekonomika.be

FILE TYPE: Textfiles

Text Files zijn bestanden die bestaan uit een sequentie van tekens, opgesplitst in lijnen van variabele lengte

Var f: Text (**File Of** Char)

Speciale procedures en functies:

ReadIn(f) Springt naar begint v/e nieuwe lijn
WriteIn(f) Voegt op het einde v/d tekst een break toe (nieuwe lijn)
EoIn(f) Duidt aan of het einde v/d lijn bereikt is

Men kan een textfile kopiëren

```
Reset(InputFile);
Rewrite(OutputFile);
While Not Eof(InputFile) Do
        Begin
                While Not EoIn(InputFile) Do
                        Begin
                          Read(InputFile, Ch);          -- Inlezen
                          Write(OutputFile, Ch)         -- Kopiëren
                        End;
                Readin(InputFile);                      -- Nieuwe lijn starten
                WriteIn(OutputFile)                     -- Nieuwe lijn sluiten
        End.
```

FILE TYPE: External Files

ZIE PAGINA 61

STRINGS

String kan een *vaste* of een *variabele* lengte hebben en is een sequentie van tekens

STRINGS: Vaste lengte

String type = **Packed Array[**1..N**] Of** Char -- *Met N groter dan 1*
! Het gaat hier om een vaste lengte (1-N)

Overzicht operaties:

- Toekenning: de lengte moet gelijk zijn van elke toekenning

Voorbeeld:

```
Var Naam1, Naam2: Packed Array[1..5] Of Char;
        Naam1 := 'Piet ';                       -- Piet is slechts 4 char, daarom spatie
        Naam2 := 'Jan  ';                       -- Jan is slechts 3 char, daarom 2x spatie
```

- Relationele operaties: ordering hangt af v/d onderliggende tekenset
- Uitvoer via WRITE-Statement: # uit te schrijven tekens kan aangegeven worden

www.ekonomika.be Dekenstraat 2, 3000 Leuven - info@ekonomika.be

STRINGS: Variabele lengte

Optie 1

Maximale lengte: N

Maar werkelijke lengte: **N** - spaties

Wel nadelig, spaties kunnen niet meer gezet worden

Optie 2

```
const maxlengte = ...;
type sting =
        record
                inhoud: packed array[1..maxlengte] of Char;
                lengte: 0..maxlengte
        end.
```

Optie 3

Linked list van tekens (onbeperkte lengte)

Wel **nadelen:**

- Omslachtig
- Onverenigbaar met char
- In en uitvoer

STRINGS: String Extensions

TURBO TURBO TURBO PASCAL PAGINA 63

3.6 Acties (opdrachten) **| Pagina 64 - 70**

Enkelvoudige opdrachten

- Toewijzingen
- Invoer- en uitvoerprocedures

Gestructureerde opdrachten

- SEQUENTIE (**Begin ...; ...; ... End**)

Compound statement

- SELECTIE (**If** ... **Then** ... **Else** ...)

If statement

Case statement

- ITERATIE (**While ... Do ...**)

While statement

Repeat Statement

For Statement

www.ekonomika.be Dekenstraat 2, 3000 Leuven - info@ekonomika.be

ENKEL VOUDIGE STATEMENTS: Toewijzingen

Toewijzingen komen tot stand in volgende vorm:

variabele := expressie

ENKEL VOUDIGE STATEMENTS: In- en uitvoerprocedures

We kunnen data inlezen via **read** en **readln**

Read(x)	Leest de input bij x
Read(x, y, ...)	Leest de input bij x, y, ... sequentieel
Readln	Volgende regel laten inlezen (**readln (input)**)
Readln(x, ...)	

We kunnen data wegschrijven via **write** en **writeln**

Write(x)	Schrijf de waarde naar uitvoerstroom
Write(x, y, ...)	Schrijft de waarden naar uitvoerstroom
Writeln	Schrijft de regel naar uitvoerstroom
Writeln(x, ...)	

GESTRUCTUREERDE OPDRACHTEN: Selecties (If Statement)

Het gaat hier om selectieprocedures v/d vorm:

if expression **then** statement

if expression **then** statement **else** statement

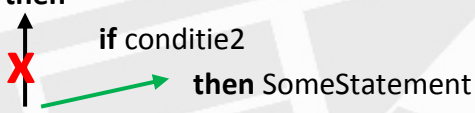
Kenmerken:

- Expressies zijn BOOLEAN
- Meerdere statements worden aangegeven via **begin** ... **end** en gescheiden door **;**
- **!!** Else verwijst steeds naar laatst open then

Voorbeeld:

if conditie1

then



Else SomeStatement --> misleidend!

www.ekonomika.be Dekenstraat 2, 3000 Leuven - info@ekonomika.be

GESTRUCTUREERDE OPDRACHTEN: Selecties (Case Statement)

Soms selectie uit meerdere statements, afhankelijk v/d ordinale expressie

! Alle mogelijkheden, constanten, voor de expressie moet men opnemen

Voor de laatste tak kan men eventueel **else** gebruiken

Voorbeeld:

```
Case Maand Of Januari, Maart, Mei, Juli, Augustus, October, December: Dagen := 31;
              April, Juni, September, November : Dagen := 30;
              Februari: If (Jaar Mod 4 = 0) And ...
                    then Dagen := 29
                    else Dagen := 28
End.
```

GESTRUCTUREERDE OPDRACHTEN: Iteraties (While Statement)

While is de meeste algemene vorm en wordt gevolgd is van de vorm:

```
while BooleanExpression do Statement
```

Wanneer de expressie "**true**" is, voert men de statement opnieuw uit

Voorbeeld:

```
I := 1
While I <= 9 Do
      Begin
        WriteIn(I);
        I := I + 2
      End
```

Eigenschappen:

- Het aantal iteraties is 0 of meer
  (*de expressie wordt getest VOOR uitvoering*)
- Het aantal iteraties is onvoorspelbaar (niet vastgelegd)
  (*de expressie wordt steeds opnieuw berekend worden*)
- Meerdere te itereren statements worden aangeduid via **Begin...End**
- De BooleanExpressie heeft de waarde **FALSE** na uitvoering v/h While Statement

ZIE DUIDELIJKE AFBEELDING PAGINA 67

www.ekonomika.be Dekenstraat 2, 3000 Leuven - info@ekonomika.be

GESTRUCTUREERDE OPDRACHTEN: Iteraties (Repeat Statements)

Speciale vorm van iteratie, minimmum aantal herhalingen is **1**, van de vorm:

Repeat StatementSequence **Until** BooleanExpression

Men controleert steeds iets, wanneer dit waar blijkt te zijn, stopt de iteratie

Voorbeeld:

```
Repeat
    Write('Geef een positief getal: ');
    Readln(GeheelGetal)
Until GeheelGetal >=0
```

Men blijft de vraag herhaling tot de gebruiker effectief een positief getal ingeeft

Eigenschappen

- Het aantal iteraties is 1 of meer (de test is nl. na de eerste iteratie)
- Het aantal iteraties is onvoorspelbaar (steeds opnieuw berekening
- Meerdere te itereren statements hebben geen nood aan **begin**...**end**
- Na de uitvoering v/d repeat tatement heeft de expressie de waarde **TRUE**

ZIE DUIDELIJKE AFBEELDING PAGINA 68

GESTRUCTUREERDE OPDRACHTEN: Iteraties (For Statement)

Het aantal iteraties is **0** of meer, maar **ligt vast bij aanvang** v/h for-statement
De waarde v/d loop-variabele is onbepaald, na de uitvoering v/h fot-statement

De expressie is van de vorm:

for variabele := start **to** stop **do** statement | Geen uitvoering als start>stop
for variabele := start **downto** stop **do** statement | Geen uitvoering als stop>start

! Meerdere statements aangeven met **begin** en **end** en gescheiden door **;**

Voorbeeld:

```
for i := a to b do
    begin
      statement;
      ...
      statement
    end
```

GESTRUCTUREERDE OPDRACHTEN: Iteraties (For Statement) VERALGEMENING

For i := Start **To** Stop **Do** S | *Met Start en Stop als Integer*

Bekijk voorbeelden 69 - 70 (alternatieven)

www.ekonomika.be Dekenstraat 2, 3000 Leuven - info@ekonomika.be

3.7 Abstractie: Procedures en Functies | Pagina 70 - 89

Procedures: opsplitsing taak in deeltaken
Vb: verfijningen, modules, stappen, subprogramma's, ...
Waarom:
- Modulariteit, leesbaarheid, onderhoudbaarheid
- Meervoudig gebruik code
Vorm:

```
procedure klasse.actie;
  begin
        Operaties en Manipulaties (statements)
  end;
```

! Declaraties zijn lokaal (enkel binnen éénzelfde procedure) - lokale parameters
! Binnen één procedure kunnen meerdere procedures voorkomen

Voorbeeld:

```
Program RijOptelling (Input, Output);                  -- Declaratie programma naam

const     Max = 10;                                    -- Declaratie van constanten
type      Index = 1..Max;                              -- Verzameling afbakenen
          Rij = array[Index] of Integer;               -- Rij declareren als opvolging
var       A, B, C: Rij;                                -- Verschillende rijen declareren

Procedure LeesRij (var R: Rij);                        -- Procedure met lokale var van type rij
          var i: index;                                -- lokale var van type index (1-10)
        Begin
          for i := 1 to Max do read(R[i])              -- Iteratie opdracht tot lezen
        End;
Procedure TelOp;
          var i: index;                                -- lokale var van type index (1-10)
        Begin
          for i := 1 to Max do C[i] := A[i] + B[i]     -- Iteratie opdracht tot sommatie
        End;
Procedure DrukRij (R : Rij)                            -- ?? Waarom geen VAR ??
          var i: index;                                -- lokale var van type index (1-10)
        Begin
          for i := 1 to Max do write(R[i])             -- Iteratie opdracht tot schrijven
        End;
Begin
        LeesRij(A);                                    -- Ingeven rij A
        LeesRij(B);                                    -- Ingeven rij B
        TelOp;                                         -- Rij A en B optellen tot C
        DrukRij(C                                      -- C weergeven
End.
```


www.ekonomika.be Dekenstraat 2, 3000 Leuven - info@ekonomika.be

Namen, binding en geheugen

Variabelen met naam verwijzen naar geheugenplaats
--> bevat specifieke waarde
Type bepaald welke waarden toegelaten zijn
(bijgevolg ook operaties)

Koppeling tussen *naam, type, referentie (adres) en waarde*
Er zijn **parameters** om de **kwaliteit v/e programmeertaal** te bepalen
- **Binding time:** tijdstijp binding verschillende elementen
Bij compilatie
Bij loading
Bij binnenkomst subprogramma
Bij uitvoering statements

Naam-declaratie binding: Scope v/d variabele
Binding time: ***bij compilatie***

Variabelen eerst gedeclareerd, daarna gebruik naam
--> Naam in statemenent hoort bij declaratie

Voorbeeld:

```
program BindingExample;
        var x: integer;
procedure A;
        begin
          write(x)
        end;
procedure B;
  var x: real;
        begin
          A;
        end;
begin
        B; A;
end.
```

Binding tussen naam en declaratie reeds bij ***compilatie*** | PASCAL
We noemen dit **static scope** | Binding is niet afhankelijk van uitvoeringsvolgorde
! Namen in bepaalde blok, zijn lokaal
! Namen in omsluitende blok, zijn globaal (voor die blok)
Via static scope is **type checking** mogelijk tijdens compilatie
reeds controle bij compilatie en niet in run-time

www.ekonomika.be Dekenstraat 2, 3000 Leuven - info@ekonomika.be

Dynamic scope (of dynamische binding) | Binding afhankelijk van uitvoeringsvolgorde
Alles wordt uigesteld tot **run-time**
Bij dynamic scope zou x eerst real zijn, daarna integer (bij uitvoeren b)
Zal leiden tot **meer flexibiliteit**, maar **geen compilatiecontrole**

Declaratie-referentie binding: Levensduur van een variabele

Levensduur: extent v/e variabele
--> *tijd dat een variabele tijdens de uitvoering v/e programma aan een geheugenplaats is toegewezen*

Bij het binnenkomen v/e blok: toekennen bij declaratie v/e variabele
! Blok wordt meerdere malen opgeroepen: declaratie kan anleiding geven tot verschillende geheugenplaatsen
Conclusie: **declaration-reference** binding gebeurt bij **run time** (binnenkomen)
Door de declaratie bij run-time kan men recursie toepassen
Wel **probleem:**
Waarde v/e variabele worden niet bewaard tussen oproepen

In sommige talen kan dit wel via **statische lokale variabelen**
- Normale scope
- Declaration-reference ***bij loading***

! Scope en levensduur moeten niet identiek zijn
Scope: *declaratie bij compilatie*
Levensduur: *declaratie bij run-time (normaal)*

Referentie-waarde binding: Vormen

Voorbeeld: **y := y + 1**

De eerste y verwijst naar de *geheugenplaats* voor opslag
De tweede y verwjst naar de inhoud v/d huidige y-waarde

Er zijn drie bindingen:
- Name-declaration binding
- Declaration-reference binding
- Reference-value binding

Als we de waarde willen zoeke ngebreuken we **dereferencing**

Name-value binding v/e **constante** gebeurd bij **compilatie**
--> kan niet gewijzigd worden in statements, geen reference nodig
Name-value binding v/e **variabele** gebeurd in de uitvoeringv/e **statement**

www.ekonomika.be Dekenstraat 2, 3000 Leuven - info@ekonomika.be

Dynamische variabelen

Variabelen zijn gekoppeld aan geheugenplaatsen zodra en zolang een blok bestaat
Pointer variabelen: variabelen waarvan waarde verwijst naar referentie
--> Declaration-reference in statement-gedeelte, men noemt dit **dynamic storage allocation**

Na een procedure is er nog een **dynamische variabele** die bestaat (maar ontoegankelijk)
--> nood aan *manuele of automatische garbage collection*

Block structure and scope

Declaraties (*constanten, types, variabelen, procedures* en *functies*) zijn lokaal in de blok

De levensduur v/e lokale variabele: start procedure - einde procedure
! Daarbuiten geen definitie
Waarom:

- Abstractie enkel significant binnen procedure
- Vermijden ongewilde neveneffecten door gebruik zelfde namen
- Verhogen leesbaarheid en onderhoudbaarheid
- Grote vrijheid in naamkeuze
- Recursie
- Geheugenbesparing door variabele geheugenbeslag

Parameters van procedures

Value parameters (call by value): uitsluitend input
--> Waarde v/d actuele parameters (variabele, constante, expressie) gekopieerd naar functie/procedure

Voorbeeld:

```
Procedure PutSpaces (aantal: posint);      end;
Function Min(i, j: integer): integer;      end;

PutSpaces(width)
...Min (1, 2)
...Min (x, y)
...Min(a, Min(b, c))
```


www.ekonomika.be Dekenstraat 2, 3000 Leuven - info@ekonomika.be

Variabele parameters (call by reference): input, transput en output
--> Naam v/d actuele parameters (variabele, array, structuur) wordt synoniem met formal parameters, beiden refereren één geheugenplaats. De referentie wordt naar de variabele doorgegeven (supplementair op de waarde).

```
Procedure Leesgetal (var Getal: integer);    end;
Procedure Order (var x, y: integer);         end;
Procedure Drukmatrix (var m: matrix);        end;

Leesgetal(Hoogte)
Order(a, b)
Order(Element[i], Element[j])
```

Functies

Functies zijn benoemede programmadelen die een enkelvoudige waarde berekenen en aanduiden

Vorm:
function klasse.actie: type
! Resultaat v/e functie is altijd één enkele waarde, moet toegekend worden bij uitvoering

Voorbeeld I:

```
Program KleinsteGemiddelde (Input, Output);      -- Programma definiëren
  Const Max = ...;                               -- Constanten declareren
  Type Index = 1..Max;                           -- Variabelen declareren
       Rij = array[index] of Integer;
  Var  A, B: Rij;

Function Min(x, y: Real): REAL;                  -- Plaatselijke variabelen
  Begin
       If x<y then min := x else min := y
  End;
Function Gemiddelde (R: Rij): Real;
       var i: Index;
           Som: real;
  Begin
       Som := 0.0;
       for i := 1 to Max do
       Som := Som + R[i];
       Gemiddelde := Som / Max                  -- Result berekenen
  End;
```


www.ekonomika.be Dekenstraat 2, 3000 Leuven - info@ekonomika.be

```
Begin
        LeesRij(A);
        LeesRij(B)
        WriteIn(Min(Gemiddelde(A), Gemiddelde(B)))
 End.
```

Voorbeeld II:

```
function TKlantenLijst.zoek_klant (naam: string): TKlant;
        var i: integer;
  begin
        result := nil;
        for i := 1 to KlantenLijst.AantalKlanten do
          if naam := KlantenLijst.Elementen[i].Naam then
                begin
                  Result := KlantenLijst.Elementen[i];
                  Break
                end;
end.
```

Iteratief sorten

ZIE VOORBEELD PAGINA 76 - 78

Recursie

ZIE CURSUS PAGNA 78 - 89 EN BIJKOMENDE BUNDEL TOLEDO VOOR VOORBEELDEN

Pagina 89 - 96 is verdere duiding programma, linked lists, etc.
(open boek)

www.ekonomika.be Dekenstraat 2, 3000 Leuven - info@ekonomika.be

Hoofdstuk 4: Units | Pagina 97 - 106

Traditionele Pascalprogramma's inhoud:

- Declaratie van globale constanten, types en variabelen
- Geheel procedures en functies (met lokale variabelen en subprocedures)
- Hoofdprogramma met procedureoproepen

Procedures en functies maken **information hiding** mogelijk
--> Men kan procedures oproepen zonder interne werking te moeten kennen
Enkel rekening nodig met *parameters* en *initialisaties*

Hergebruik reeds geschreven procedures vaak moeilijker:

- Reeds geschreven procedures moeten ingesloten worden en opnieuw gecompileerd
- Initialisaties die samenhangen met procedure of geheel, moeten in oproep zitten
- Declaratie van types en variabelen moeten toegankelijk zijn (globaal)
- Gemeenschappelijke datastructuur van samenhorende procedures is niet verbergbaar

Units vangen deze problematief op, het is een vorm, de voorloper, van **OO**s

Units en Abstracte datatypes

Unit: verzameling gerelateerde constanten, types, variabelen, procedures en functies (eventueel ook een hoofddeel met initialisaties)
! Unit is bijna een afzonderlijk programma (seperate compilation mogelijk)

Via Units kan men procedures en functies oproepen m.b.t. éénzelfde onderwerp
--> wel ondergebracht in afzonderlijke units met eigen structuren, variabelen, ... (**encapsulation**)
Inhoud is afschermbaar voor externe wereld (**information hiding**)

Structuur van een unit

Gelijkaardig aan een programma

- Declaratie constanten, types, variabelen, procedures en functies
- Nodige initialisaties opnemen

Twee grote secties:

- **Interface** | Publiek gedeelte v/h programma
- **Implementation** | Verborgen, privaat, gedeelte v/h programma

ZIE VOORBEELD PAGINA 98

www.ekonomika.be Dekenstraat 2, 3000 Leuven - info@ekonomika.be

Interface Section

Zichtbaar toegankelijk deel voor programma en units
! De uitwerking is verborgen in implementatie secties
Interface section toont beschikbare types, variabelen en procedures
(werking hiervan is onbelangrijk)

Uses clause: maakt oproepen andere units mogelijk
! Units kunnen elkaar niet wederzijds op roepen via uses clauses in interface
--> nood aan uses clause in implementatie section (oplossing voor circulatie referentie)

Implementation Section

Bjkomende declaraties kunnen toegevoegd worden, afgescherm van oproepende programma's

Het implementatiegedeelte bevat ook de **main body** van elke procedure en functie
(*deze bevinden zich in de interface*)
! Hoofdding van procedure /functie in implementatie moet identiek zijn aan die in interface

Circulatie referenties

Soms heeft de *aangeroepen unit* gegevens nodig uit de *aanroepende unit*
De uses-clausules van beide units zullen naar de andere unit moeten verwijzen
! Zal leiden tot een foutmelding: oneindige lus van unit 1 naar 2 naar 1 naar ...

Nood aan **verplaatsing van uses-statement** van één v/d units van interface naar implementatie (maakt niet uit welke, beter de tweede)

Illustratie:

```
Unit Unit 1
Interface
        Uses ..., ..., Unit2;
...
Implementation
...

Unit Unit 2
Interface
        Uses ..., ...;
...
Implementation
        Uses Unit1;
...
```


www.ekonomika.be Dekenstraat 2, 3000 Leuven - info@ekonomika.be

Initialization section | vrijblijvende sectie

Gebruikt bij initialisatie van datastructuren en variabelen die gebruikt worden door unit of deergegeven worden via interface

! Bij uitvoering programma: eerst initialisatie secties units, dan hoofdprogramma

Finalization section | vrijblijvende sectie

Stuk code dat men oproept bij beëindiging v/h hoofdprogramma
Vb: opslaan of vernietigen van gegevens

Gebruik

Programma is opgesplitst in modules
- Hoofdprogramma
- Eventuele units

Het programma start altijd met de vorm: **program** + naam v/h hoofdprogramma
Een unit start altijd met de vorm: **unit** + naam v/d unit

Units krijgen een andere extensie

Units zijn apart compileerbaar (modulariteit), voordeel:
- Groot programma opgedeeld in modules, hercompileren beperkt tot module met veranderingen
- Elke unit apart ingeladen in apart code segment (omvang niet meer beperkt tot segment)
- Units kunnen ingeladen of verwijderd worden

Korte begrippenlijst

Genericiteit

Unit met bepaalde datastructuur kan men specifiek koppelen aan datatype
Men kan gebruik maken van **untyped parameters**
bouwen van generische procedures en units onafhankelijk van type parameters

Information hiding

Men kan informatie zichtbaar houden voor gerelateerde procedures, maar verbergen voor de rest v/h programma (**private declaraties in implementatie sectie**)

Overloading

Operatoren of procedures dragen zelfde naam
Onderscheiding: op basis van operands of parameters
! NIET MOGELIJK IN STANDAARD PASCAL

www.ekonomika.be Dekenstraat 2, 3000 Leuven - info@ekonomika.be

Encapsulation

Het samenvoegen van datastructuren en bewerkingen om concrete implementatiedetails die afgescherm zijn van de buiten wereld, simpel te kunnen wijzigen

ZIE TOEPASSINGEN PAGINA 103 - 105

Voordelen: modulariteit, information hiding en encapsulation

ZIE TEKST 105 - 106

www.ekonomika.be Dekenstraat 2, 3000 Leuven - info@ekonomika.be