Programmeren – Blok A

http://www.win.tue.nl/~wstomv/edu/2ip05/

College 5

Tom Verhoeff

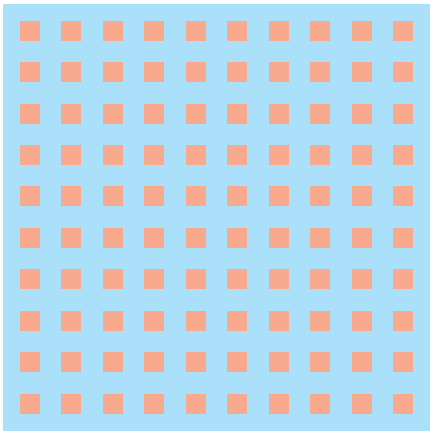
Technische Universiteit Eindhoven
Faculteit Wiskunde en Informatica
Software Engineering & Technology

Opmerkingen aan T.Verhoeff@TUE.NL

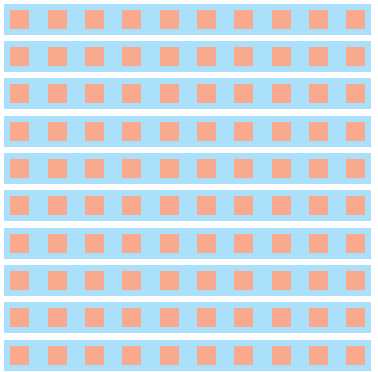
Onderwerpen

- 2-dimensionale arrays
- Enumeratie type
- Set type
- 2-dimensionale keuze

2-dimensionale arrays



2-dimensionale arrays via 1-dimensionale arrays



Geneste arrays in Pascal

var a: **array** [*Char*] **of** **array** [*Boolean*] **of** *Integer* ;

a ['A'] [*False*] := 13

Verkort:

var a: **array** [*Char*, *Boolean*] **of** *Integer*;

$a['A', False] := 13$

2-dimensionale arrays in Pascal

const
  NRijen = ...; { aantal rijen, $1 \leq NRijen$ }
  NKolommen = ...; { aantal kolommen, $1 \leq NKolommen$ }

type
  TRijIndex = 0 .. *NRijen* − 1;
  TKolomIndex = 0 .. *NKolommen* − 1;
  TRij = **array** [*TKolomIndex*] **of** *Integer*;
  TMatrix = **array** [*TRijIndex*] **of** *TRij*;

Verkort:

TMatrix = **array** [*TRijIndex*, *TKolomIndex*] **of** *Integer*;

2-dimensionale arrays 'uitpakken' in Pascal

var a: *TMatrix*; $\begin{pmatrix} a[0] \\ a[1] \\ \vdots \\ a[NRijen-1] \end{pmatrix}$

TRijIndex ↓	*TKolomIndex* → 0	1	...	*j*	...	*NKolommen* − 1
0	$a[0,0]$	$a[0,1]$	...	$a[0,j]$	...	$a[0,\ldots]$
1	$a[1,0]$	$a[1,1]$	...	$a[1,j]$	...	$a[1,\ldots]$
⋮	⋮	⋮		⋮		
i	$a[i,0]$	$a[i,1]$	...	$a[i,j]$	...	$a[i,\ldots]$
⋮	⋮	⋮		⋮		
NRijen−1	$a[\ldots,0]$	$a[\ldots,1]$	...	$a[\ldots,j]$	...	$a[\ldots,\ldots]$

2-dimensionaal array inlezen

var
  rij: *TRijIndex*; { doorloopt rijen }
  kol: *TKolomIndex*; { doorloopt kolommen }

...

  for *rij* := 0 **to** *NRijen* − 1 **do begin**
    { lees de volgende rij }

    for *kol* := 0 **to** *NKolommen* − 1 **do begin**
      read (*mat* [*rij*, *kol*])
    end { for *kol* }

  ; *readln*
  end { for *rij* }

2-dimensionaal array schrijven

```
const
  KolomScheiding = ' ';  { i.v.m. afdrukken }
  KolomBreedte = 2;  { i.v.m. afdrukken, ≥ 1 }

...

  for rij := 0 to NRijen − 1 do begin
    { schrijf de volgende rij }

    for kol := 0 to NKolommen − 1 do begin
      write ( KolomScheiding, mat [ rij, kol ] : KolomBreedte )
    end { for kol }

  ; writeln
  end { for rij }
```


2-dimensionaal array 'random' initialiseren

```
const
  NElementen = 100;  { aantal elementwaarden, ≥ 1 }

type
  TElement = 0 .. NElementen − 1;  { de array elementen }

...

  Randomize    { initialiseer random generator }

; for rij := 0 to NRijen − 1 do begin
    { initialiseer de volgende rij }

    for kol := 0 to NKolommen − 1 do begin
      mat [ rij, kol ] := Random ( NElementen )
    end { for kol }

  end { for rij }
```


Probleem: Energiepillen

Beschouw deze 3×4 matrix:

0	1	30	5
2	10	0	3
4	20	7	99

Wat is de maximale padsom bij enige wandeling van linksboven naar rechtsonder, waarbij alleen naar rechts en naar onder gestapt wordt?

(Hoeveel van dergelijke wandelingen zijn er?)

Energiepillen: analyse

Er zijn $\binom{3+4-2}{3-1} = \binom{5}{2} = 10$ wandelingen mogelijk.

Van de 5 stappen moet je er 2 naar onder kiezen en 3 naar rechts.

0	1	30	5
2	10	0	3
4	20	7	99

De maximale padsom bedraagt hier 138.

In 20×20 matrix: $\binom{38}{19} \approx$ 35 miljard wandelingen!

Het enumeratie-type

(⟨*naam*⟩ , . . . , ⟨*naam*⟩)

Waardenverzameling: de verzameling van opgesomde namen

Operaties: constantes ⟨*naam*⟩, totale ordening ($<, =, >$), *succ*, *pred*, conversie naar *Integer* via *ord*

Voorgedefinieerd enumeratie-type:

type *Boolean* = (*False*, *True*);

$False < True$ *succ* (*False*) = *True* *pred* (*True*) = *False*

ord (*False*) = 0 *ord* (*True*) = 1

Waarschuwing m.b.t. ordening op *Boolean*

a	b	a <= b
False	False	True
False	True	True
True	False	False
True	True	True

Uit de tabel blijkt dat `a <= b` logisch correspondeert met $a \Rightarrow b$.

Het enumeratie type: een vergelijking

const
Red = 0;
Green = 1;
Blue = 2;

type
PrimaryColor = *Red* .. *Blue*;

type
PrimaryColor = (*Red*, *Green*, *Blue*);

Het tweede biedt betere bescherming tegen fouten.

Het eerste laat toe $Red + Green$, het tweede niet.

Bij het tweede hoef je zelf geen volgnummers te verzinnen.

Het set-type

set of ⟨*scalair type*⟩

Waardenverzameling: alle deelverzamelingen van ⟨*scalair type*⟩

Operaties: Beschouw expressies v, w: **set of** T; s, t: T;

- Constantes: [e_1 , . . . , e_N] met $e_i \in T$ of e_i = s .. t
- Binaire operatoren: $+$ (vereniging), $-$ (verschil), $*$ (doorsnede)
- Binaire relaties: t **in** v (element van), $v \leq w$ (deelverz. van)

Het set-type: een vergelijking

var a, b: **array** [*PrimaryColor*] **of** *Boolean*; c: *PrimaryColor*;	**var** v, w: **set of** *PrimaryColor*; c: *PrimaryColor*;
for $c :=$ *Red* **to** *Blue* **do** $a[c] :=$ *False* **for** $c :=$ *Red* **to** *Blue* **do** $a[c] :=$ *True*	$v := [\,]$ $v :=$ [*Red* .. *Blue*]
a [c] a [c] $:=$ *True* a [c] $:=$ *False*	c **in** v $v := v +$ [c] $v := v -$ [c]
for $c :=$ *Red* **to** *Blue* **do** $a[c] := a[c]$ **and** $b[c]$ . . . ? . . . . . . ? . . .	$v := v * w$ $v \neq [\,]$ $v \leq w$

Toepassing van set-constante

```
1 var
2   teken: Char; { gelezen teken }
3 ...
4   if ( ( '0' <= teken ) and ( teken <= '9' ) )
5     or ( teken = ' ' ) or ( teken = '-' ) then ...
```

```
1 const
2   GeldigTeken = [ '0' .. '9', ' ', '-' ]; { de geldige tekens }
3
4 var
5   teken: Char; { gelezen teken }
6 ...
7   if a in GeldigTeken then ...
```


Gevalsonderscheid: meer alternatieven, genest

Kwadratische vergelijking $ax^2 + bx + c = 0$ oplossen met abc-formule

Discriminant $D = b^2 - 4ac$

1. $a = 0$: geen kwadratische vergelijking

   . . .

2. $a \neq 0$: een echte kwadratische vergelijking

   (a) $D < 0$: geen reële oplossingen

   (b) $D = 0$: één reële oplossing

   (c) $D > 0$: twee reële oplossingen

Geneste keuze en meerwegkeuze in Pascal

```
1   D := sqr ( b ) - 4 * a * c { discriminant }
2 ; if a = 0 then begin { 1. niet kwadratisch }
3     ...
4   end
5   else begin {@ a <> 0, dus 2. echt kwadratisch }
6     if D < 0 then begin { 2.(a) geen reele oplossingen }
7       ...
8     end
9     else if D = 0 then begin { 2.(b) een reele oplossing }
10      ...
11    end
12    else begin {@ D > 0, 2.(c) twee reele oplossingen }
13      ...
14    end { else }
15  end { else }
```

Indentering past zo niet bij gevalsonderscheid

```
1   (* ZO NIET DOEN *)
2 ; if a = 0 then begin { niet kwadratisch }
3     ...
4   end
5   else if D < 0 then begin { geen reele oplossingen }
6     ...
7   end
8   else if D = 0 then begin { een reele oplossing }
9     ...
10  end
11  else begin {@ D > 0, twee reele oplossingen }
12    ...
13  end { else }
```


Indentering past zo ook niet bij gevalsonderscheid

```
1   (* ZO NIET DOEN *)
2 ; if a = 0 then begin { niet kwadratisch }
3     ...
4   end
5   else begin
6     if D < 0 then begin { geen reele oplossingen }
7       ...
8     end
9     else begin
10      if D = 0 then begin { een reele oplossing }
11        ...
12      end
13      else begin {@ D > 0, twee reele oplossingen }
14        ...
15      end { else }
16    end { else }
17  end { else }
```


2-dimensionale selectie: waarom

Bijv. het percentage commentaar in een Pascal programma bepalen

Iets over gelezen tekens onthouden (maar niet alles): toestand

Net gelezen teken: invoer

Combinatie van toestand en invoer bepaalt actie en nieuwe toestand

	invoer →			
↓ toestand	`''''`	`'{'`	`'}'`	anders
*Outside**	*InString*	*InComment*		
InString	*Outside*			
InComment			*Outside*	

*Begintoestand

2-dimensionale selectie: hoe

```
1 var
2   toestand: ( Outside, InString, InComment ); { classificatie }
3   teken: Char; { doorloopt tekens in tekstbestand }
4
5 ...
6
7   { initialisatie }
8   toestand := Outside
```

2-dimensionale selectie: hoe

```
1     { verwerk teken }
2   ; case toestand of
3       Outside: begin
4         case teken of
5           '''': ... toestand := InString ...
6           '{': ... toestand := InComment ...
7         else { toestand ongewijzigd }
8         end { case teken }
9       end; { Outside }
10
11      InString: begin
12        if teken = '''' then ... toestand := Outside ...
13      end; { InString }
14
15      InComment: begin
16        if teken = '}' then ... toestand := Outside ...
17      end { InComment }
18    end { case toestand }
```


De case-opdracht

case ⟨*formule van scalair type* T⟩ **of** ⟨*lijst gevallen*⟩ **end**

waarbij ⟨*lijst gevallen*⟩ een lijst van 'gevallen' is, elk van de vorm

e_1 , ... , e_N : ⟨*opdracht*⟩ ;

met $e_i \in T$ of $e_i =$ s .. t voor $s, t \in T$

optioneel kan direct vóór **end** nog opgenomen worden

else ⟨*lijst opdrachten*⟩

Deeltentamen 2XP05: stof, materiaal

- Elementair Object Pascal: **program**, **var**, *Integer*, *Boolean*, *Char*, *String*, *Real*, **begin**, **end**, ';', ':=', *readln*, *writeln*, **if**, **then**, **else**, **while**, **do**, **array**, **for**, **repeat**, **until**, tekstbestanden, ...
- Lazarus, o.i.d.
- FreePascal documentatie
- Slides, samenvatting Pascal, voorbeelden, eigen programma's

Brilliance

Michael Jackson

Software Requirements & Specifications:
a lexicon of practice, principles and prejudices

Addison-Wesley, 1995

ISBN 0–201–87712–0

Brilliance

Some years ago I spent a week giving an in-house program design course at a manufacturing company in the mid-west of the United States.

On the Friday afternoon it was all over.

The DP Manager, who had arranged the course and was paying for it out of his budget, asked me into his office.

Brilliance

'What do you think?' he asked.

He was asking me to tell him my impressions of his operation and his staff.

'Pretty good,' I said. 'You've got some good people there.'

Program design courses are hard work; I was very tired; and staff evaluation consultancy is charged extra.

Anyway, I knew he really wanted to tell me his own thoughts.

Brilliance

'What did you think of Fred?' he asked. 'We all think Fred is brilliant.'

'He's very clever,' I said. 'He's not very enthusiastic about methods, but he knows a lot about programming.'

'Yes,' said the DP Manager. He swivelled round in his chair to face a huge flowchart stuck to the wall: about five large sheets of line printer paper, maybe two hundred symbols, hundreds of connecting lines.

Brilliance

'Fred did that. It's the build-up of gross pay for our weekly payroll. No one else except Fred understands it.'

His voice dropped to a reverent hush.

'Fred tells me that he's not sure he understands it himself.'

'Terrific,' I mumbled respectfully. I got the picture clearly.

Fred as Frankenstein, Fred the brilliant creator of the uncontrollable monster flowchart.

That matched my own impression of Fred very well.

'But what about Jane?' I said. 'I thought Jane was very good. She picked up the program design ideas very fast.'

'Yes,' said the DP Manager. 'Jane came to us with a great reputation. We thought she was going to be as brilliant as Fred. But she hasn't really proved herself yet.

We've given her a few problems that we thought were going to be really tough, but when she finished it turned out they weren't really difficult at all. Most of them turned out pretty simple.

She hasn't really proved herself yet — if you see what I mean?'

I saw what he meant.

(End of quote)