Programmeren – Blok B

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

College 10

Tom Verhoeff

Technische Universiteit Eindhoven
Faculteit Wiskunde en Informatica
Software Engineering & Technology

Opmerkingen aan T.Verhoeff@TUE.NL

Onderwerpen

- Lazarus tips
- Modularisatie met behulp van het **unit**-mechanisme
- Test drivers, automatisch uitvoeren van testgevallen
- Efficiëntie van programma's

Lazarus Tips

- Schakel de debugger uit
- Zorg voor dezelfde instellingen als in peach^3:

  Why does my program crash in peach^3, but not when I run it on my laptop?

- Zie: www.win.tue.nl/~wstomv/edu/lazarus/faq.html

Modularisatie in Object Pascal met het unit-mechanisme

Hergebruik van code door copy/paste hindert toekomstig onderhoud, omdat alle kopieën afzonderlijk aangepast moeten worden.

Een **unit** bundelt een stel definities (van constantes, types, routines).

Een **unit** is in een apart bestand ondergebracht.

Een **unit** kan eenvoudig als eenheid worden (her)gebruikt in allerlei programma's, zonder copy/paste.

Een **unit** kan separaat vertaald en gecontroleerd (getest) worden.

Syntax van een unit in eenvoudigste vorm

unit ⟨*unitnaam*⟩ ;
  { ⟨*Auteur*⟩, ⟨*id.nr.*⟩, ⟨*datum*⟩ }
  { ⟨*Samenvatting van doel*⟩ }

interface

⟨*publieke-definities-en-routine-specificaties*⟩

implementation

⟨*privé-definities-en-routine-implementaties*⟩

end.

Voorbeeld van een unit: het interface

```
1  unit Histograms;
2    { (c) 2008, Tom Verhoeff }
3    { Voorzieningen voor histogrammen }
4
5  interface
6
7  const
8    MaxHistSize = 1000;
9
10 type
11   THistIndex = 0 .. MaxHistSize - 1;
12
13   THistogram = record
14     size: 0 .. MaxHistSize;
15     freq: array [ THistIndex ] of Cardinal;
16       { freq[i] is gedefinieerd als 0 <= i < size }
17   end;
```


Voorbeeld van een unit: het interface (vervolg)

```
1  procedure InitHistogram ( out h: THistogram; n: Integer );
2    { pre:  0 <= n <= MaxHistIndex
3      post: h.size = n /\ (A i: 0 <= i < n: h.freq[i] = 0) }
4
5  procedure TurfInHistogram ( var h: THistogram; i: THistIndex; m: Integer );
6    { pre:  0 <= i < h.size  ==>  0 <= h.freq[i] + m
7      post: h = old h,
8            behalve dat h.freq[i] = old h.freq[i] + m als 0 <= i < h.size }
9
10 function TotaalInHistogram ( const h: THistogram ): Cardinal;
11   { pre: True
12     ret: (+ i: 0 <= i < h.size: h.freq[i]) }
13
14 procedure WriteHistogram ( var f: TextFile; const h: THistogram );
15   { pre:  f is geopend voor schrijven
16     post: een frequentietabel van h is geschreven naar f }
```


Voorbeeld van een unit: de implementation

```
1  implementation
2
3  procedure InitHistogram ( out h: THistogram; n: Integer );
4  var
5    i: THistIndex; { doorloopt h.freq }
6  begin
7    with h do begin
8      size := n
9
10   ; for i := 0 to n - 1 do begin
11       freq [ i ] := 0
12     end { for i }
13
14   end { with h }
15 end;
16
17 ...
18 end.
```

Gebruik van een unit

```
 1 program Dobbelen;
 2   { Simuleer
 3     dobbelspel. }
 4
 5 uses
 6   Histograms;
 7
 8 { definities en
 9   declaraties }
10 ...
11
12 begin
13   ...
14 end.
```

```
 1 unit Histograms;
 2   { Voorzieningen voor histogrammen }
 3
 4 interface
 5 { publieke definities en
 6   routine specificaties }
 7 ...
 8
 9 implementation
10 { prive definities en
11   routine implementaties }
12 ...
13
14 end.
```


Gebruik van een unit (2)

In het gebruikende programma worden

op de plaats van **uses** Histograms;

alle publieke definities en routine specificaties opgenomen,

alsof of ze daar ter plekke stonden.

unit Lists: publieke constantes en types

```
 1 unit Lists; { Voorzieningen voor werken met eenvoudige lijsten }
 2
 3 interface
 4
 5 const
 6   MaxListLen = 24; { maximum lengte van een lijst, 1 <= MaxListLen }
 7   MinEntry = 0; { kleinste waarde in een lijst }
 8   MaxEntry = 99; { grootste waarde in een lijst, MinEntry <= MaxEntry }
 9
10 type
11   TIndex = 0 .. MaxListLen - 1; { index in een lijst }
12   TEntry = MinEntry .. MaxEntry; { waarde in een lijst }
13   TList = record
14     len: 0 .. MaxListLen; { actuele lengte van de lijst }
15     val: array [ TIndex ] of TEntry; { de rij van waarden in de lijst }
16       { de lijst bestaat uit de waarden val[0] t/m val[len-1] }
17   end;
18   { Bij s: TList moet s.val[i] gedefinieerd zijn voor 0 <= i < s.len }
```


unit Lists: specificaties van routines

```
 1 procedure WriteList ( var f: Text; const ident: String; const s: TList );
 2   { pre:  f is geopend om te schrijven
 3     post: lijst s is geschreven naar f, volgens het formaat
 4           ident.len: ...
 5           ident.val: ... }
 6
 7 procedure GenerateList ( out s: TList; const n, m, d, c: Integer );
 8   { genereer lijst s met lengte n volgens patroon m, d, c:
 9     blokken van lengte d met gelijke waarden c, c+m, c+2m, ... }
10   { pre:  0 <= n <= MaxListLen,  0 < d,
11           MinEntry <= m * (i div d) + c <= MaxEntry  voor 0 <= i < n
12     post: s.len = n /\ (A i: 0 <= i < n: s.val[i] = m*(i div d) + c)  }
13
14 procedure Find ( const s: TList; const x: TEntry;
15                  out found: Boolean; out pos: TIndex );
16   { pre:  (A i, j: 0 <= i < j < s.len: s.val[i] <= s.val[j])
17     post: found = (E i: 0 <= i < s.len: s.val[i] = x) /\
18           (found  ==>  0 <= pos < s.len  /\  s.val[pos] = x) }
```

Hoe de implementatie van Find te testen?

Je weet: de implementatie gebruikt een zogenaamde Binary Search.

Hoe de functionele correctheid van Find (automatisch) testen?

We laten testen op efficiëntie, robuustheid en gebruikersvriendelijkheid even buiten beschouwing.

Maak een test driver: een apart programma dat

- unit Lists gebruikt,
- geschikte (slimme) aanroepen doet van Find,
- de resultaten van Find *onafhankelijk* controleert, en
- de testuitslagen rapporteert.

Functionaliteit van Find testen: testgevallen kiezen

1. Test met lijsten met lengtes
   - 0 t/m 5 (want die hebben allemaal iets bijzonders),
   - enkele typische grotere waarden,
   - `MaxListLen - 1` en `MaxListLen`
2. Test met "kam"lijsten `s` waarvoor `s.val[i] = 2*i + 1` en zoek `x = 0, 1, 2, ..., 2*s.len`, dan weet je het resultaat
3. Test met lijsten met alle waarden gelijk (bijv. 0)
4. Test met lijsten met enkele gelijke waarden

A Test Driver

program *ListsTestDriver*;
{ Test *Find* in unit *Lists* and write defect log to standard output }

uses
Lists; { unit under test }

procedure *TestFind2* (n: *Integer*);
...

var
n: *Integer*; { list length };

begin
for $n := 0$ **to** 5 **do** *TestFind2* (n)
end.

A Test Case

procedure *TestFind2* (n: *Integer*);
{ **pre** $0 \leq n \leq MaxListLen$
post Tested *Find* for "comb" list with length n
Defect log written to standard output }
var
s: *List*; { input for *Find* }
x: *Integer*; { input for *Find* }
found: *Boolean*; { output from *Find* }
pos: *Integer*; { output from *Find* }

begin { *TestFind2* }
writeln (`'TestFind: n='`, n)
; *GenerateList*(s, n, 2, 1, 1)
{@ $s.len = n \wedge (\forall i : 0 \leq i < n : s.val[i] = 2i + 1)$ (a "comb" list) }

A Test Case

```
; for x := 0 to 2*n do begin
    Find ( s, x, pos, found ) { routine under test }
  ; if found ≠ odd(x) then begin
      writeln ( 'found error: x = ', x, 'found = ', found )
    end
    else begin { found OK }
      if found then begin
        if (pos < 0) or (n ≤ pos) or (s.val[pos] ≠ x) then begin
          writeln ( 'pos error: x = ', x, 'pos = ', pos )
        end { if }
      end { if }
    end { else }
  end { for x }

end; { TestFind2 }
```


Efficiëntie van programma's

- Tijd voor uitvoeren van de opdrachten
- Geheugen voor opslaan van de variabelen

Machines worden steeds sneller en geheugens steeds groter.

Waarom is efficiëntie (juist) dan relevant?

Rekentijd als functie van probleemomvang N

Onderscheid: worst-case, average-case, best-case, amortized

Naam	**Rekentijd**	**Extra bij kN**
constant	C	$\ldots + 0$
logaritmisch	$C \log N$	$\ldots + C \log k$
lineair	CN	$\ldots * k$
(lineairitmisch)	$CN \log N$	$\ldots * k + (k \log k)CN$
kwadratisch	CN^2	$\ldots * k^2$
kubisch	CN^3	$\ldots * k^3$
exponentieel	$C2^N$	$\ldots * 2^{(k-1)N}$

Hoeveel meer kun je doen met $2\times$ snellere machine?

Stel $N = 1000$ kost 1 uur op machine A.
Hoeveel verder kom je op een $2\times$ snellere machine B in 1 uur?

	Rekentijd	**N op A**	**N op B**	**Extra op B**
logaritmisch	$\log_2 N$	1000	10^6	999 000
lineair	N	1000	2000	1000
(lineairitmisch)	$N \log_2 N$	1000	1844	844
kwadratisch	N^2	1000	1414	414
kubisch	N^3	1000	1260	260
exponentieel	2^N	1000	1001	1

Standaard zoekalgoritmen

Bounded Linear Search

```
1   pos := 0
2
3 ; for i := 1 to n do begin
4     if Voldoet(i) then pos := i
5   end { for i }
```

Linear Search

```
1   {@ pre: (E i: 0 <= i: Voldoet(i) ) }
2   pos := 0
3
4 ; while not Voldoet(pos) do begin
5     pos := pos + 1
6   end { while }
```

Bounded Linear Search with early termination

```
1   pos := 0
2 ; i := n
3
4 ; while pos <> j do begin
5     if Voldoet(pos) then i := pos
6     else pos := pos + 1
7   end { while }
```


Meer standaardalgoritmen

Binary Search

Sorteren

Kortste pad in graaf vinden

. . .

Efficiëntie van KeizerKiezer

Eenvoudig programma behandeld in college 3

Onderwerp van de KeizerKiezer Prijsvraag

Quiz over efficiëntie

Gegeven zijn de definities:

const
  $Size$ = ...; { grootte van array a ($0 \leq Size$) }

var
  a: **array** [0 .. $Size - 1$] **of** $Char$;
  n: $Integer$; { bovengrens (invoer, $0 \leq n$) }
  i, j: $Integer$; { om a te doorlopen }

Rekentijd: Bepaal hoe vaak (als functie van n) de procedure $write$ wordt aangeroepen in onderstaande programmafragmenten.

Geheugengebruik: Bepaal hoe groot $Size$ minimaal moet zijn.

Quiz: Tabel invullen

	Rekentijd aantal *write*	Geheugen *Size*
1		
2		
3		
4		
5		

Als functie van n

Quiz: Fragement 1

```
  i := 0

; while i ≠ n do begin
    write ( a[i] )
  ; i := i + 1
  end { while }
```


Quiz: Fragement 2

```
  i := 0
; j := 0

; while i ≠ n do begin
    write ( a[i + j + 1] )
  ; j := (j + 1) mod n
  ; i := i + ord(j = 0)
  end { while }
```

N.B. $ord(False) = 0$ en $ord(True) = 1$.

Quiz: Fragement 3

```
  i := 0
; j := n

; while i ≠ n do begin
    i := i + 1

  ; while sqr(i) + sqr(j) > sqr(n) do begin
      write ( a[i * n − j] )
    ; j := j − 1
    end { while }

  end { while }
```

N.B. $sqr(n) = n^2$

Quiz: Fragement 4

```
  i := 1

; while i < n do begin
      write ( i )
   ; i := 2 * i
  end { while }
```


Quiz: Fragement 5

```
{ pre: (∀ i : 0 ≤ i ≤ n : a[i] = '0' ) }
repeat
   i := 0

 ; while a[i] = '1' do begin
      a[i] := '0'
   ; i := i + 1
   end { while }

 ; a[i] := '1'
 ; write ( a[0] )
until i = n
```

Extra: Hoe vaak wordt $i := i+1$ uitgevoerd?

Quiz: Ingevulde tabel

	Rekentijd aantal *write*	Geheugen *Size*
1	n	n
2	n^2	$2n$
3	n	n^2
4	$\lceil \log_2 n \rceil$	0
5	2^n	$n+1$

Ad 5. $i := i+1$ wordt *gemiddeld* 2 keer uitgevoerd per slag door de repeat, dus in totaal iets van 2^{n+1}. De binnenlus verandert dus niet wezenlijk iets aan de rekentijd.

Niet behandelde aspecten van 'moderne' programma's

- Object-Orientation: 2IP05, Blok C
- Grafische User Interfaces: 2IP05, Blok C
- Exception handling: 2IP05, Blok C
- Koppeling met externe database voor beheer van persistente data
- Koppeling met netwerk, gedistribueerde (client–server) software
- Multi-threading voor parallelle verwerking

Terugblik

Computer als programmeerbare automaat: toestand en verandering

Programmeertaal Pascal, programmeergereedschap Lazarus

Probleemstelling, specificatie, ontwerp, code

Correctheid, opmaak, commentaar, naamgeving, opdeling

Syntax, semantiek, pragmatiek

Complexiteit beteugelen door 'verdeel en heers'

Specificatie van deelprobleem, maken en gebruiken van deeloplossingen; aanroep en implementatie relateren via contract

Terugblik

Programmastructuur voor lezen en schrijven van bestanden

'Verdeel en heers' toepassen op gegevens

Systematisch testen

Efficiënt tijd- en geheugengebruik

Variabele, naam, waarde, type, expressie, const, array, opdracht (read, write, toekenning, if, while, for, repeat, case, with), tekstbestand, procedure, function, parameter (const, out, var, value), lokaal–globaal, record