Pascal uitgediept
Data structuren

MSX Computer & Club Magazine nummer 68-juni/juli 1994
Herman Post

Scanned, ocr'ed and converted to PDF by HansO, 2001

In deze aflevering wordt bekeken hoe zelf een datastructuur kan worden gemaakt en gebruikt. Het nadenken over de structuur van de data en een manier om deze goed te kunnen gebruiken wordt ook uit de doeken gedaan.
Het is in Pascal erg gemakkelijk om met een bepaald type variabele te werken. Zowel de standaard typen als de zelfge-definieerde typen zijn eenvoudig en goed te gebruiken. Soms komt het echter voor dat het gebruik van een standaard type nogal veel data vraagt en daarmee te veel computergeheugen claimt. Ik wil dit demonstreren aan de hand van twee programma's. Niet alleen de manier van werken, maar ook het idee achter de gekozen structuur wil ik uitleggen.

Tekstscroll

Het eerste voorbeeld is een tekstscroll op scherm 0. Eerst even de probleemomschrijving:
Lees een tekstfile vanuit DOS in en plaats deze op het scherm. Als de tekst te groot is om op l scherm te plaatsen, zorg dan dat er met de cursortoetsen zowel voor- als achteruit door de tekst heen kan worden gelopen.
Op het eerste gezicht lijkt dit een vrij eenvoudige opgave, maar als we er wat over gaan nadenken duiken er wat probleempjes op. Om een regel uit de tekst op te slaan in het geheugen is een string de meest voor de hand liggende methode. Om de volledige tekst op te kunnen slaan is een array van strings dan een logische oplossing. Het probleem is dan dat we moeten weten uit hoeveel regels de tekst is opgebouwd en daarnaast hoe lang iedere string moet zijn om de langste regel te kunnen inlezen. We kunnen dit oplossen door eenvoudig de regellengte op 255 te plaatsen. Hiermee kunnen we dan iedere ASCII-tekst inlezen, maar er wel is erg veel loze opslagruimte in de tekst. ledere regel is dan immers 255 karakters lang, ook als de regel maar 5 karakters bevat.
In het programma is dan ook voor een andere manier gekozen. De tekst wordt opgeslagen in een buffer van karakters, waarbij een variabele aangeeft waar in de buffer op dat moment de string staat die moet worden weergegeven. Daarmee zijn de meeste problemen opgelost: we kunnen immers alle strings weer uit de buffer lezen. Omdat alle strings een variabele lengte hebben is het nu niet meer mogelijk om direct naar het begin van een regel te springen. Om echter ook door de tekst terug te scrollen is het nodig om naar het begin van de vorige regel te gaan. Daarom wordt, nadat de string in de buffer is geplaatst, ook de lengte van die string er nog even achter gezet. Nu is het teveel aan opslagruimte al weggewerkt, blijft alleen nog het probleempje bij het weergeven over. Op het scherm kunnen 24 regels staan. Deze regels kunnen maar 80 karakters lang zijn en de regels die we inlezen kunnen wel 255 karakters lang zijn. Daarom worden de regels tijdens het inlezen gecontroleerd op hun lengte en als ze te lang zijn even opgedeeld in meerdere regels. De tekst staat nu in het geheugen met de juiste maximale regellengte en zonder allemaal loze ruimte. Bovendien kan er zonder problemen naar boven en naar beneden door de tekst gewandeld worden. Voor dit laatste worden er wat variabelen bijgehouden die aangeven welke de eerste en de laatste regel op het scherm zijn, hoeveel regels er in het geheugen staan en wat de laatste gebruikt plaats in de karakterrij is.

```
PROGRAM TekstScrol;

CONST maxmem = 3000;

TYPE str255 = STRING[255];

VAR rij         : ARRAY[0..maxmem] OF CHAR;
    regel       : str255;
    startpos,
    eindpos,
    regelaantal,
    maxpos      : INTEGER;
    tel,lengte  : BYTE;
    k           : CHAR;
    file1       : TEXT;

PROCEDURE PointerDown(VAR pointer:INTEGER);
BEGIN
  pointer:=pointer-BYTE(rij[pointer-1])-2
END;

PROCEDURE PointerUP(VAR pointer:INTEGER);
BEGIN
  pointer:=pointer+BYTE(rij[pointer])+2
END;

PROCEDURE BouwScherm;
VAR y : BYTE;
BEGIN
  y:=0;
  CLRSCR;
  eindpos:=startpos;
  REPEAT
    MOVE(rij[eindpos],regel,80);
    GOTOXY(1,SUCC(y));
    WRITE(regel);
    y:=SUCC(y);
    PointerUp(eindpos)
  UNTIL (y=24) OR (eindpos>=maxpos)
END;

PROCEDURE ScrolUp;
BEGIN
  IF startpos=0 THEN EXIT;
  PointerDown(startpos);
  PointerDown(eindpos);
  MOVE(rij[startpos],regel,80);
  GOTOXY(1,1);
  INSLINE; {eerst naar beneden scrollen}
  WRITE(regel)
END;

PROCEDURE ScrolDown;
BEGIN
  IF eindpos=maxpos THEN EXIT;
  MOVE(rij[eindpos],regel,80);
  GOTOXY(1,24);
  WRITELN; {eerst omhoog scrollen}
  WRITE(regel);
  PointerUp(startpos);
```

```
  PointerUp(eindpos)
END;

PROCEDURE CharOut(k:CHAR);
BEGIN
  INLINE($3A/K/$FD/$2A/$C0/$FC/$DD/
         $21/$A2/$00/$CD/$1C/$00)
END;

BEGIN
  CONOUTPTR:=ADDR(CharOut);
  ASSIGN(file1,'scrol.pas');
  RESET(file1);
  startpos:=0;
  eindpos:=0;
  regelaantal:=0;
  maxpos:=0;
  REPEAT
    READLN(file1,regel);
    REPEAT
      regelaantal:=SUCC(regelaantal);
      lengte:=LENGTH(regel);
      IF lengte>79 THEN lengte:=79;
      IF maxpos+lengte>maxmem THEN
      BEGIN
        CLRSCR;
        WRITELN('textfile is te groot, maximaal ',maxmem,' bytes
groot!');
        HALT;
      END;
      FOR tel:=0 TO lengte DO rij[tel+maxpos]:=regel[tel];
      IF LENGTH(regel)>79 THEN rij[maxpos]:=#79;
      maxpos:=maxpos+lengte+2;
      rij[maxpos-1]:=CHAR(lengte);
      DELETE(regel,1,80);
    UNTIL regel='';
  UNTIL EOF(file1);
  CLOSE(file1);
  BouwScherm;
  REPEAT
    READ(KBD,k);
    IF k=#30 THEN ScrolUp;   {cursor omhoog}
    IF k=#31 THEN ScrolDown; {cursor omlaag}
  UNTIL (k=#27) OR (k=#32);  {esc of spatie}
  CLRSCR
END.
```

Priemgetallen

Het tweede programma dat ik wil bespreken gaat over priemgetallen. Eerst weer even de probleemomschrijving:
Eereken alle priemgetallen tot 50000 met behulp van de zeef van Eratosthenes, en geef deze op het scherm weer.
Om dit programma te kunnen uitleggen is het natuurlijk nodig om te weten wat priemgetallen zijn en om te weten hoe de zeef werkt. Priemgetallen zijn getallen die precies twee delers hebben. Uit deze definitie volgt dat het getal l zelf geen priemgetal is. Bij sommige wiskundige berekeningen gaat men er vanuit dat ook negatieve getallen priem kunnen zijn, maar dat laat ik hier buiten beschouwing omdat meestal alleen de positieve getallen beschouwd worden. De zeef van Eratosthenes is een berekening waarbij eerst een rij getallen genomen wordt van 2-n. Het kleinste getal uit deze rij is een priemgetal. Nu worden alle getallen uit de tafel van dit laagste getal weggestreept en opnieuw is het kleinste getal een priemgetal. Dit gaat door totdat de hele rij is weggestreept. Tot zover de omschrijving en uitleg van het gestelde probleem. Nu de oplossing en ook nu lijkt alles weer eenvoudiger dan dat het is. Om namelijk een rij getallen van 2 tot 50000 in het geheugen op te nemen zouden we al 50000 integers nodig hebben, wat zou neerkomen op 100000 bytes of-
wel zo'n 97 kB. Dit is op ons computertje veel te veel. We zouden kunnen zeggen dat we alleen de oneven getallen opslaan, omdat deze er na het eerste gevonden priemgetal toch al uitgeveegd worden, maar zelfs dan is er nog zo'n 48 kB aan data nodig. Dit werkt dus niet echt handig, al houden we het idee van die oneven getallen wel vast omdat het erg veel data scheelt. Van de getallen die we moeten opslaan hoeven we echter niet de waarde te hebben, we willen alleen onthouden of ze al of niet in de zeef zitten. En hier ligt dan ook het antwoord van het probleem. In een SET wordt aangegeven of een getal wel of niet in de set voorkomt. Het getal zelf wordt dus niet opgeslagen. Kijk eventueel nog eens in de vorige aflevering van Pascal uitgediept. In een set van 32 bytes kunnen we voor 256 getallen aangeven of ze wel of niet voorkomen. Als we nu 50000 delen door 256 dan blijkt dat we aan 196 sets voldoende hebben. Denken we nu ook nog even aan de opslag van alleen de oneven getallen, dan valt de helft weg en hebben we aan 98 sets voldoende om alle getallen op te slaan. Dit kan natuurlijk weer prachtig in een array van sets. Aan het begin worden er twee van deze arrays gedeclareerd: één voor de zeef en één voor de gevonden priemgetallen. De rest van het programma is vrij duidelijk en goed te begrijpen. Alleen het weergeven van de getallen boven MAXINT heb ik even buiten beschouwing gelaten. Dit is één van de leuke addertjes, die in dit Pascal-gras voorkomen.
Het is namelijk niet mogelijk om een getal groter dan MAXINT met een WRITE op het scherm te plaatsen. Zodra getallen groter worden dan MAXINT, worden ze direct als negatief gezien door de compiler. Dit maakt voor het berekenen niets uit, maar bij het weergeven krijgen we wat vreemde resultaten. Nu weet u meteen waarom ik hier alleen positieve priemgetallen bereken. De gebruikte methode is niet bijzonder netjes of snel, maar zet alle negatieve integers wel goed—positief—op het scherm.

```
{$R+}
PROGRAM priem50000;

CONST setgroote =256;
      maxelement=255;
      setdelen=97;

TYPE natuurlijk = 0..MAXINT;

VAR zeef,priemen      : ARRAY[0..setdelen] OF SET OF 0..maxelement;
    volgpriem         : RECORD
                          deel,element : natuurlijk;
                        END;
    veelvoud,nwepriem,
    p,n,teller        : natuurlijk;  { zou ook INTEGER mogen zijn}
    leeg              : BOOLEAN;
    getal             : INTEGER;     { mag niet NATUURLIJK zijn}

PROCEDURE WriteNeg(w : INTEGER);
BEGIN
  w:=w-30000;
  WRITE(w DIV 10000+3:2,
        w MOD 10000 DIV 1000:1,
        w MOD 1000 DIV 100:1,
        w MOD 100 DIV 10:1,
        w MOD 10:1);
END;

BEGIN
  { toekennen beginwaarden }
  FOR p:=0 TO setdelen DO
  BEGIN
    zeef[p]:=[0..maxelement];
    priemen[p]:=[]
  END;
  zeef[0]:=zeef[0]-[0];
  leeg:=FALSE;
  volgpriem.deel:=0;
  volgpriem.element:=1;

  { zoek volgende priem }
  WRITELN('De priemgetallen worden berekend ');
  WITH volgpriem DO
  REPEAT
    WHILE NOT (element IN zeef[deel]) DO element:=SUCC(element);
    priemen[deel]:=priemen[deel]+[element];
    nwepriem:=SUCC(element SHL 1);
    veelvoud:=element;
    p:=deel;
    WHILE p<=setdelen DO
    BEGIN
      zeef[p]:=zeef[p]-[veelvoud];
      p:=p+(deel SHL 1);
      veelvoud:=veelvoud+nwepriem;
      WHILE veelvoud>maxelement DO
      BEGIN
        p:=SUCC(p);
        veelvoud:=veelvoud-setgroote
      END
    END;
    IF zeef[deel]=[] THEN
```

```
    BEGIN
      leeg:=true;
      element:=0
    END;
    WHILE leeg AND (deel<setdelen) DO
    BEGIN
      deel:=deel+1;
      leeg:=zeef[deel]=[]
    END
  UNTIL leeg;

  { weergeven van de priemgetallen }
  WRITELN('En hier zijn ze dan:');
  teller:=0;
  FOR p:=0 TO setdelen DO
    FOR n:=0 TO maxelement DO
      IF n IN priemen[p] THEN
      BEGIN
        getal:=2*n+1+p*setgroote*2;
        IF getal<0 THEN WriteNeg(getal) ELSE WRITE(getal:6);
        teller:=SUCC(teller);
        IF (teller MOD 8)=0 THEN WRITELN;
      END;
END.
```

Dataopslag primair

U hebt natuurlijk al begrepen dat deze aflevering niet bedoeld is om u te leren hoe u een tekst over het scherm kunt laten scrollen of om u bij te brengen wat priemgetallen zijn. Het gaat erom dat u nadenkt over de manier van data opslag binnen het programma en dat u daarbij ook bekijkt of een niet direct voor de hand liggende methode misschien beter geschikt is. In het geval van de tekstscroll zou er teveel ruimte verloren zijn gegaan en bij de priemgetallen zou de berekening niet eens mogelijk zijn als de eenvoudigste en meest voor de hand liggende methode werd gebruikt. U begrijpt het alweer, proberen, fouten maken, opnieuw proberen, en... weer betere Pascal programma's is uw beloning.