14. Sorteren

14.1 Inleiding

Bij het beheren van gegevens zal het vaak nodig zijn om deze gegevens te sorteren. Als we dat niet zouden doen, zou het wel erg moeilijk kunnen worden om een bepaald gegeven terug te vinden. Een adressenbestand wordt vaak op naam en/of postcode gesorteerd, en een rij getallen op oplopende of aflopende volgorde. Een programmeur kan niet zonder enige basiskennis van sorteertechnieken. Bij sorteren is er altijd sprake van een rij gegevens. Dit kunnen eenvoudige of zeer ingewikkelde gegevenstypen zijn. Als het om een samengesteld gegevenstype zoals een record gaat, is er altijd minstens één sleutelveld waarop de sorteerroutine wordt losgelaten.

Voor een computer is sorteren een relatief tijdrovend karwei. Er bestaat een groot aantal verschillende sorteermethoden, van zeer eenvoudig tot buitengewoon ingewikkeld. De verwerkingstijd van de diverse sorteermethoden verschilt nogal. In dit hoofdstuk worden twee sorteermethoden uitgebreid behandeld.

14.2 BubbleSort

BubbleSort is een trage sorteermethode die eigenlijk alleen gebruikt wordt om kleine hoeveelheden gegevens te sorteren. In het programma SORT_1.PAS staat een getypeerde constante van het type Array [0..9] of Word. Deze array bevat tien elementen waarbij elk element een cijfer bevat uit de reeks 0 tot en met 9.

SORT_1.PAS moet deze cijfers in oplopende volgorde sorteren en daarna de array op het scherm zetten:

```
PROGRAM SORT_1;
USES CRT, DIVERSEN;

CONST
   RIJ: Array[0..9] of Word = (0,8,7,6,5,9,1,2,3,4);

VAR
   I, OPSLAG: Word;
   KLAAR    : Boolean;

BEGIN
   REPEAT
      KLAAR := True;
      FOR I := 0 TO 8 DO
      BEGIN
         IF RIJ[I] > RIJ[I+1] THEN
         BEGIN
            KLAAR    := False;
            OPSLAG   := RIJ[I+1];
            RIJ[I+1] := RIJ[I];
            RIJ[I]   := OPSLAG
         END
      END
    UNTIL KLAAR;
    ClrScr;
    FOR I := 0 TO 9 DO Writeln(RIJ[I]);
    Readln
END.
```

Regels:Toelichting:

[1]3-4 Declareer een Array [0..9] of Word als getypeerde constante.
[2]5-7 Declareer de benodigde variabelen.
[3]9-21Ga een REPEAT-lus in die aangehouden wordt tot KLAAR de waarde True heeft.
[4]11-20Ga een FOR-lus in waarin steeds twee elementen van de array bekeken worden. Als de waarde in RIJ[I] groter is dan de waarde in RIJ[I+1], dan worden beide elementen verwisseld. Telkens als een verwisseling plaatsvindt, wordt de variabele KLAAR op False gezet.
[5]22-24Veeg het scherm schoon en zet met behulp van een FOR-lus het resultaat van de sortering op het scherm.

Toelichting:

[1]De variabele RIJ wordt gedeclareerd als getypeerde constante en gevuld met cijfers uit de reeks 0 tot en met 9. De cijfers staan in een willekeurige volgorde.

[2]De variabele I wordt gebruikt als index en de variabele OPSLAG dient tijdelijk als opvang voor de waarde in één van de elementen in RIJ. KLAAR is een variabele van het type Boolean. Deze variabele krijgt de waarde True als er niets meer te sorteren valt.

[3]De REPEAT-lus wordt aangehouden tot de variabele KLAAR de waarde True heeft. Deze variabele wordt bij iedere doorloop van de lus op True gezet.

[4]Nadat KLAAR op True gezet is, gaan we een FOR-lus in die net zo vaak doorlopen wordt als er elementen in de array staan min 1. In de lus wordt telkens van een tweetal elementen bekeken of de één groter is dan de ander. Als dit het geval is, worden beide elementen verwisseld door RIJ[I+1] in de variabele OPSLAG te zetten. Daarna krijgt RIJ[I+1] de waarde van RIJ[I], en krijgt RIJ[I] de waarde die nu in OPSLAG staat. De waarden in de twee elementen zijn nu verwisseld.

Hiermee is ook verklaard waarom er één element afgetrokken moet worden bij het definiëren van de FOR-lus. Als we dat niet zouden doen, dan zou het programma RIJ[9] met RIJ[10] willen vergelijken. RIJ[10] bestaat echter niet, omdat we een Array [0..9] hebben gedeclareerd. Zonder de correctie zouden we dus

buiten de grenzen van de array treden.

De werking van de boolean KLAAR moet nu ook duidelijk zijn. Bij iedere doorloop van de REPEAT-lus wordt hij True. Als er elementen verwisseld zijn, wordt KLAAR False. Als bij het verlaten van de FOR-lus de boolean KLAAR op True staat, is er dus geen enkel element verwisseld. Als er niets meer te verwisselen valt, zijn we klaar met sorteren.

[5]Als RIJ nu op het scherm gezet wordt, zien we dat de cijfers keurig in oplopende volgorde gesorteerd zijn. Als je met de ingebouwde debugger (zie bijlage 3) de gang door de lussen volgt en de inhoud van de elementen en de verwisseling daarvan bestudeert, kun je zien waar de BubbleSort-methode haar naam aan heeft ontleent. Tijdens de sortering komen de elementen als een soort belletjes bovendrijven.

14.3 QuickSort

De code voor de sorteermethode QuickSort wordt door Borland meegeleverd. Een van de voorbeeldprogramma’s is QSORT.PAS dat 1000 willekeurige getallen razendsnel sorteert. QuickSort is een recursieve sorteermethode en beschikt over een lokale procedure Sort. QuickSort roept deze procedure zelf één keer aan. Sort roept daarna zichzelf zo vaak aan als nodig is om de gegevens te sorteren.

Recursieve programma’s zijn niet makkelijk te doorgronden. Bij de bestudering kan het gebeuren dat je wordt bekropen door wanhoopsgevoelens en gedachten als: "Hoe zit dit nu precies in elkaar?"

Om de sorteermethode QuickSort goed te kunnen begrijpen, is de door Borland geleverde code enigszins aangepast. Met name de benaming van de variabelen zijn gewijzigd, en ook sorteert het programma niet 1000 willekeurige getallen, maar hetzelfde aantal getallen als het voorgaande programmaatje SORT_1.

Met behulp van de toelichting kun je het sorteerproces stap voor stap volgen. Maak er geen probleem van als je soms de weg kwijtraakt. Ook als je het niet helemaal kunt doorgronden, kun je QuickSort evengoed in je programma’s gebruiken:

PROGRAM SORT_2;

```
USES CRT, PRINTER;

CONST
   MAX = 10;

TYPE
   Lijst = Array[1..MAX] of Integer;

CONST
   Gegevens: Lijst = (0,8,7,6,5,9,1,2,3,4);

VAR
   I: Integer;

PROCEDURE QuickSort(VAR A: Lijst; Laag, Hoog: Integer);

PROCEDURE Sort(Links, Rechts: Integer);
VAR
   L, R, X, Y: Integer;

BEGIN
   L := Links;
   R := Rechts;
   X := A[(L+R) DIV 2];
   REPEAT
      WHILE A[L] < X DO Inc(L);
      WHILE X < A[R] DO Dec(R);
      IF L <= R THEN

      BEGIN
         Y    := A[L];
         A[L] := A[R];
         A[R] := Y;
         Inc(L);
         Dec(R)
      END;
   UNTIL L > R;
   IF Links < R THEN Sort(Links, R);
   IF L < Rechts THEN Sort(L, Rechts)
END;

BEGIN
   Sort(Laag,Hoog)
END;
```

```
BEGIN
   ClrScr;
   QuickSort(Gegevens, 1, MAX);
   Writeln;
   FOR I := 1 TO MAX DO Write(Gegevens[I]:3);
   Readln
END.
```

Regels:Toelichting:

3-4Declareer een constante MAX voor het aangeven van het maximum.
5-6Declareer het type Lijst als een Array [1..MAX] of Integer.
7-8Declareer een getypeerde constante van het type Lijst.
9-10 Declareer een integer voor index-doeleinden.
11-36Procedure QuickSort(VAR A:Lijst;Laag,Hoog:Integer).
12-33Lokale procedure van QuickSort.
Procedure Sort(Links,Rechts:Integer).
13-14Declareer de lokale variabelen van Sort.
[3]16-17Zet L op de waarde die doorgekregen is in Links, en zet R op de waarde van Rechts.
[4]18Geef X de waarde die in het element van A staat, waarvan de index berekend wordt door de formule "(Links + Rechts) DIV 2".
[5]19-30 Ga een REPEAT-lus in die verlaten wordt als L groter is dan R.
[6]20Ga WHILE-lus in waarin L verhoogd wordt tot er een element gevonden wordt dat groter of gelijk is aan X.

[7]21Ga een WHILE-lus in waarin R verlaagd wordt tot er een element gevonden wordt dat kleiner of gelijk is aan X.
[8]22-29Als L kleiner of gelijk is aan R, verwissel dan de waarden in de elementen aangegeven door L en R. Verhoog L en verlaag R.
[9]30Toets of L groter is dan R. Als dit niet het geval is, keer dan terug naar het begin van de REPEAT-lus.
[10]31Als Links kleiner is dan R, roep dan de procedure Sort recursief aan en gebruik Links en R als argument.
[11]32Als L kleiner is dan Rechts, roep dan Sort recursief aan en gebruik L en Rechts als argument.
[2]34-36Roep vanuit de procedure QuickSort zijn lokale procedure Sort aan. Gebruik Laag en Hoog als argument.
37-43Hoofdprogramma.
[1]39 Roep QuickSort aan.
40-42 Laat het resultaat van de sortering op het scherm zien.

Toelichting:

[1]Er wordt een array van tien elementen naar QuickSort gestuurd. In deze array staan de cijfers 1 tot en met 10 (1 tot en met MAX) in een willekeurige volgorde. Bijvoorbeeld:

[0] [8] [7] [6] [5] [9] [1] [2] [3] [4]

[2]De procedure QuickSort roept de lokale procedure Sort aan.

[3]De waarden van Links en Rechts worden respectievelijk in L en R gezet. De variabelen L en R worden gebruikt om de afmeting van een groep aan te geven.

[4]De lokale variabele X krijgt de waarde van het element waarvan de index berekend wordt met "(Links + Rechts) DIV 2". In dit geval is dat het vijfde element. X wordt dus 5.

[5]Er wordt nu een REPEAT-lus ingegaan, die aangehouden wordt tot de parameter Links groter of gelijk is aan R, en L groter of gelijk is aan de parameter Rechts.

[6]In een WHILE-lus wordt L verhoogd tot er een element gevonden is met een waarde die groter of gelijk is aan X. Uitgaande van de array zal L de waarde 2 krijgen.

[7]De lokale variabele R wordt in een WHILE-lus verlaagd tot er een element is dat kleiner of gelijk is aan X. R blijft gewoon 10, omdat A[10] kleiner is dan X.

[8]Omdat L kleiner is dan R, worden de gevonden elementen met elkaar verwisseld. De variabele A kun je je nu zo voorstellen:

[0] [4] [7] [6] [5] [9] [1] [2] [3] [8]

[9]De variabele L wordt nu verhoogd en R verlaagd. Er wordt gekeken of na deze bewerking L misschien groter is dan R. In dat geval wordt de REPEAT-lus verlaten. Als L groter is dan R, bevinden zich aan de linkerzijde van R geen getallen meer die groter of gelijk zijn aan X.

De variabele L is niet groter dan R, dus gaan we terug naar het begin van de REPEAT-lus. In de twee WHILE-lussen wordt weer gezocht naar te verwisselen elementen. De elementen 3 en 9 worden gevonden. Deze worden verwisseld:

[0] [4] [3] [6] [5] [9] [1] [2] [7] [8]

Na de bewerking heeft L de waarde vier en R acht. Omdat L nog steeds niet groter is, worden in het begin van de REPEAT-lus de twee WHILE-lussen weer ingegaan. Vanaf positie vier wordt een element gezocht dat groter is dan X (die was 5). Het eerste element dat gevonden wordt is element vier. R blijft op acht staan:

[0] [4] [3] [2] [5] [9] [1] [6] [7] [8]

In de WHILE-lus blijft L vijf. A[5] is immers gelijk aan X. R blijft zeven:

[0] [4] [3] [2] [1] [9] [5] [6] [7] [8]

L is nu zes en ook R heeft de waarde zes gekregen. De REPEAT-lus wordt verlaten.

[10]Links stond nog steeds op 1 en is dus kleiner dan R. Sort wordt dus recursief aangeroepen met Links en R als argument. Nu wordt in de array gezocht tussen element één en element vijf. X krijgt de waarde van het middelste element van dat bereik: "((1+5)/2 = 3)". X krijgt dus de waarde drie.

Het eerste element dat groter is dan X, is element twee. Het eerste element dat kleiner is dan X, is element vijf. Deze worden weer verwisseld:

[0] [1] [3] [2] [4] [9] [5] [6] [7] [8]

Na een nieuwe bewerking staat R op vier en L op drie. Ook deze elementen moeten verwisseld worden:

[0] [1] [2] [3] [4] [9] [5] [6] [7] [8]

Na de verwisseling is L groter dan R. Dat betekent dat de REPEAT-lus wordt verlaten. Omdat Links nog steeds groter is dan R, wordt Sort opnieuw aangeroepen. Het middelste element van bereik wordt bereikt met: "(1+3)/2= 2". X krijgt de waarde van A[2]. Dit is 1. Na de zoekactie hebben L en R dezelfde waarde. Zij wijzen naar hetzelfde element. Een enkel element kun je nu eenmaal niet sorteren. Er vindt geen verwisseling plaats en de REPEAT-lus wordt verlaten.

Nu blijkt dat aan de eerste voorwaarde voor het recursief aanroepen van Sort niet meer voldaan wordt, want Links is niet meer kleiner dan R.

[11]Ook aan de voorwaarde voor de tweede aanroep wordt niet voldaan, want L is niet groter dan Rechts. Zij zijn beide drie.

Omdat aan geen van de voorwaarden meer voldaan wordt, verlaten we de procedure. Nu treedt de recursie in werking. Sort was twee keer recursief aangeroepen. De waarden in de variabelen en de parameters zijn op de stack bewaard gebleven. De volgende tabel

laat zien wat die waarden waren:

Recursie:	Links:	Rechts:	L:	R:	X:	Aanroep:
1	1	10	6	5	5	1
2	1	5	4	3	3	1

Bij het verlaten van de procedure Sort was de array alleen aan de linkerzijde gesorteerd. De recursie zorgt er nu voor dat ook de rechterzijde gesorteerd wordt.

Er wordt teruggekeerd naar de opdracht direct na de recursieve aanroep. Als er een recursieve aanroep is geweest vanaf de eerste aanroepregel, dan wordt er teruggekeerd naar de tweede aanroepregel. Bij een aanroep vanaf de tweede aanroepregel wordt teruggekeerd bij END.

Eerst wordt teruggekeerd bij de derde aanroep. De gegevens hiervan liggen immers bovenop de stack. Nu wordt getoetst of L kleiner is dan Rechts. Als dit het geval is, dan moeten er wellicht nog elementen verwisseld worden. Sort wordt aangeroepen om het bereik 4...5 te onderzoeken. Hier worden verder geen elementen verwisseld. De procedure wordt verlaten en er wordt teruggekeerd naar de eerste aanroep.

Nu wordt Sort aangeroepen met de waarden zes en tien, zodat nu de rechterkant van de array gesorteerd wordt. Het eindresultaat is een serie keurig in oplopende volgorde gesorteerde getallen:

[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]

Het principe van het sorteren is als volgt: Het sorteren aan de linkerzijde van de tabel is een mechanisme dat van groot naar klein werkt door telkens het bereik te halveren. Hierdoor worden de kleinste getallen naar de linkerzijde gedreven. Als dit gebeurd is, volgt er een recursieve bewerking. Deze bewerking is het spiegelbeeld van de eerste bewerking. Deze werkt van klein naar groot, met als resultaat dat de grote getallen naar rechts gedreven worden.

Het verschil in snelheid tussen BubbleSort en QuickSort kan worden gemeten door het aantal verwisselingen te tellen. BubbleSort maakt voor het sorteren van de array van tien elementen 26 verwisselingen. QuickSort heeft voor dezelfde array maar 12 verwisselingen nodig.

249

249

249

258

258

258