Programmatielogica

Sorteren

Webleren

School je gratis bij via het internet. Waar en wanneer je wilt.

www.vdab.be/webleren

18/05/2015

Inhoud

Hoofdstuk 1. Inleiding

1.1. Algemene informatie

1.1.1. Waarom?

Waarom leren programmeren? De belangrijkste redenen op een rijtje:

- Programmeren leert je een probleem-oplossende manier van denken aan. Je leert **analytisch denken.**
- Als je kan programmeren kun je een hoop dingen **automatiseren** en ben je waarschijnlijk ook handig met andere ict-gerelateerde zaken.
- Je leert werken met **gegevens**. Informatica is een studie over gegevens en informatie. Bij programmeren leer je hoe je met gegevens om moet gaan en wat je ermee kunt doen.
- Websites maken is waardevol tegenwoordig. Bijna elk bedrijf of project heeft een **bijhorende site** die moet worden onderhouden (Javascript, PHP,…).
- …

In deze cursus **programmatielogica** leer je gestructureerd programmeren, dwz dat je leert zelfstandig problemen op te lossen met de computer. Je gebruikt Nassi-Shneiderman diagrammen. Deze leggen de basis voor het echte programmeerwerk in één of andere taal.

De cursus programmaticalogica bestaat uit verschillende delen.
Basisstructuren, **Procedures en functies** en **Arrays en Strings** heb je reeds doorgenomen.

Dit deel gaat over het nut van sorteren en bespreekt de verschillende sorteermethodes. De meeste programmeertalen hebben al ingebouwde sorteermodules, maar toch is het goed om een inzicht te hebben in wat hier achter kan zitten.

Als je na dit onderdeel nog zin hebt in meer, kan je je nog verder verdiepen in **Bestanden**.

De nadruk in alle delen ligt op het probleemoplossend denken. Na het tekenen van de diagrammen gaan we deze schema's ook omzetten naar echte programmacode om ze uit te testen.

1.1.2. Werkwijze

In deze cursus gebruiken we een tekentool om Nassi-Shneiderman diagrammen, PSD's of Programma Structuur Diagrammen, te tekenen (Structorizer). Je kan alle diagrammen ook maken met pen en papier. We gebruiken ook het programma Lazarus om onze code uit te testen. Later meer over beide programma's.

In deze cursus zijn heel wat oefeningen voorzien. Je kan ze onderverdelen in twee categorieën:

- *Gewone oefeningen:*
  Dit is het elementaire oefenmateriaal dat noodzakelijk is om de leerstof onder de knie te krijgen.
  Bij deze oefeningen kan je ook altijd een modeloplossing (van het PSD) terug vinden.
- *Opdrachten voor de coach:*
  Per hoofdstuk is er een opdracht voor de coach voorzien. Deze kan als 'test' voor dat

hoofdstuk dienen. Dit is een samenvattende oefening van de voorgaande leerstof. Voor deze opdrachten zijn er geen modeloplossingen voorzien.

Als je deze cursus volgt binnen een **competentiecentrum**, spreek je met je **instructeur** af hoe de opdrachten geëvalueerd worden.

Anders stuur je je oplossingen van de *opdrachten voor de coach* door aan je coach ter verbetering. Als je een probleem of vraag hebt over één van de gewone oefeningen, kan je ook bij je **coach** terecht. Stuur steeds zowel het .nsd-bestand als het .pas-bestand dat je gemaakt hebt door. Dit maakt het voor je coach makkelijker om je vraag snel te beantwoorden.

Let op!
Het is niet de bedoeling om met Structorizer te leren werken, wel om een probleem te leren analyseren en in een gestructureerd diagram weer te geven. Structorizer en Lazarus zijn enkel hulpmiddelen om het tekenen en uittesten van die diagrammen te vereenvoudigen.

Hoofdstuk 2. Sorteren

2.1. Inleiding

In dit onderdeel bekijken we het nut van sorteren. Om de verschillende sorteermethodes uit te leggen vertrekken we steeds vanuit dezelfde beginsituatie.

Sorteren van gegevens is een elementaire – maar daarom geen eenvoudige – vaardigheid voor de programmeur. We gaan in deze cursus wat dieper in op enkele **sorteeralgoritmen**.

Zoals je zal zien, is niet de volgorde van de sortering problematisch (m.a.w. of de gegevens van groot naar klein of omgekeerd moeten worden geordend), maar wel de manier waarop je dat zo **efficiënt mogelijk** (dus in zo weinig mogelijk tijd, in zo weinig mogelijk bewerkingen) kan doen.

Er bestaan relatief eenvoudige sorteeralgoritmen zoals de *selection sort*, de *insertion sort* en de *bubblesort*, maar er bestaan ook complexere algoritmen zoals de *quicksort*.

We zullen, in wat volgt, een aantal van deze gekende sorteeralgoritmen bekijken, telkens aan de hand van een praktisch voorbeeld.

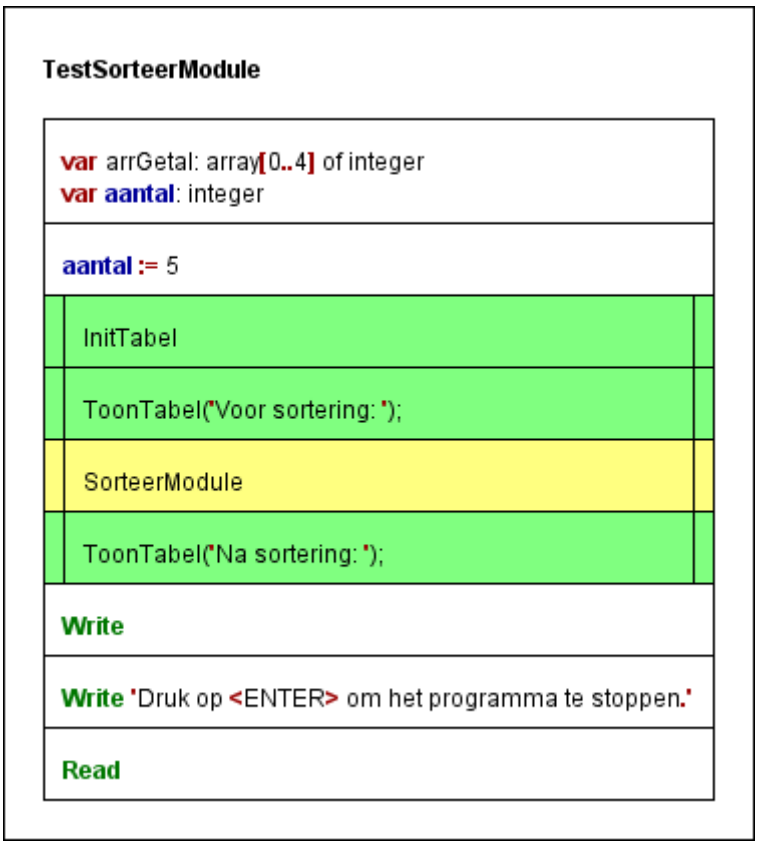
2.1.1. Voorbeeld

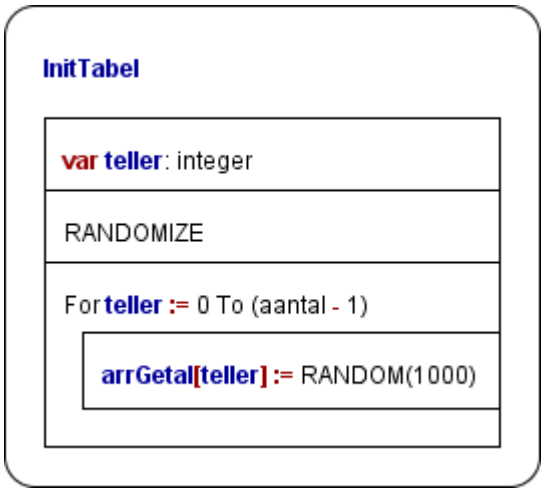
We gebruiken telkens hetzelfde voorbeeldje om kort de werking van elk algoritme uit te leggen. We baseren ons op een tabel van 5 elementen met volgende basisinhoud:

5	8	2	9	6

Ook bij het opstellen van de PSD's gaan we telkens uit van eenzelfde tabel. We gaan ervan uit dat deze tabel reeds gevuld is met willekeurige waarden.

Dit zijn de startschema's:





We focussen ons verder enkel op de sorteermodule.

2.2. Selection sort

2.2.1. Principe

De "**selection sort**" (of sortering door selectie) is een eenvoudige manier van sorteren. De werkwijze is:

- zoek het kleinste element
- zet dat element op de eerste plaats (verwissel dus het kleinste element met het element op de eerste plaats)
- zoek opnieuw het kleinste element (vanaf het tweede element) en ook dit element wordt op de juiste plaats (de tweede plaats) gezet (verwissel dus het tweede kleinste element met het element op de tweede plaats)
- herhaal deze werkwijze voor n elementen, maar telkens voor een kortere reeks waarden (je laat telkens het eerste element uit de vorige routine weg)

De bewerking stopt wanneer de tabel "op" is: je sorteert het laatste element niet mee, want als de eerste n-1 elementen op hun juiste plaats staan, staat het laatste element vanzelf goed.

2.2.2. Voorbeeld

Stel we hebben volgende tabel:

5	8	2	9	6

Het kleinste element wordt gezocht (nl. het 3de, met waarde 2), en wordt vooraan gezet (door het gevonden element te wisselen met het eerste). De tabel wordt dan:

2	8	5	9	6

Het eerste element staat nu op de juiste plaats, dus we beperken ons nu tot de laatste 4 getallen. Het kleinste daarvan is 5, op de 3de plaats (van de volledige tabel). We wisselen dus het 2de en 3de element en krijgen dan:

2	5	8	9	6

Op dezelfde manier zoeken we het kleinste element uit de laatste 3 en zetten dat op de 3de plaats:

2	5	6	9	8

En tenslotte het kleinste element uit de laatste 2 en dat zetten we op de 4de plaats:

2	5	6	8	9

Op dit moment is de tabel gesorteerd. Het laatste element staat vanzelf goed.

Merk dus op dat voor een tabel van 5 elementen er 4 sorteerdoorgangen nodig zijn.

2.2.3. Varianten

Deze werkwijze kent nog een paar varianten:

- i.p.v. telkens het kleinste element vooraan te zetten, kan je ook steeds het grootste element achteraan zetten. Je zal dan de elementen met elkaar vergelijken steeds te beginnen vanaf het eerste element, maar telkens één element minder achteraan in de tabel. Je bekomt hetzelfde resultaat, nl. een sortering van klein naar groot;
- anderzijds kan je ook altijd het kleinste element achteraan (of het grootste element vooraan) zetten. Je krijgt dan als resultaat een sortering van groot naar klein.

2.2.4. Opgave

Werk nu een procedure uit die **arrGetal** gaat sorteren via een Selection sort.

Algemeen geformuleerd ziet de werkwijze er zo uit:

- We zoeken het kleinste element uit een tabel van N elementen.
  Dit betekent:
  - We stellen dat het eerste element het “kleinste tot dan toe” is.
  - We vergelijken elk volgend element met het “kleinste tot dan toe”.
  - Indien het element kleiner is, dan wordt dat het nieuwe “kleinste tot dan toe”.
- We wisselen dat element met het eerste uit de tabel
  Hiertoe:
  - Kopieer je het eerste element naar een tijdelijke variabele.
  - Kopieer je het kleinste element naar het eerste.
  - Kopieer je de tijdelijke variabele naar de plaats waar het kleinste stond.
- We herhalen deze werkwijze voor de laatste N-1 elementen, daarna voor de laatste N-2 en zo verder, tot de laatste 2.
  Voor een tabel van N elementen, zijn er dus N-1 sorteerdoorgangen nodig.

Oplossing

2.3. Bubblesort

2.3.1. Principe

De "**bubblesort**" (of sortering door omwisseling) lijkt op de selection sort, maar is iets minder efficiënt:

- Vergelijk twee opeenvolgende elementen.
- Wissel de twee elementen als het eerste groter is dan het tweede.
- Nadat zo de hele rij doorlopen is, staat het grootste element achteraan. Het is naar boven "gebubbeld".
- Herhaal deze werkwijze telkens met één element (het laatste) minder.

Deze werkwijze is minder efficiënt omdat er per sorteerdoorgang meerdere omwisselingen kunnen gebeuren, daar waar er bij de selection sort slechts één omwisseling is per sorteerdoorgang.

2.3.2. Voorbeeld

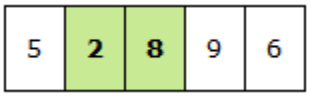
Stel we hebben volgende tabel:

5	8	2	9	6

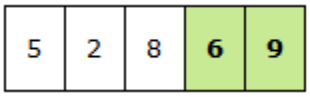
Eerste iteratie

Bij de eerste iteratie gaan we door de hele tabel, we vergelijken dus

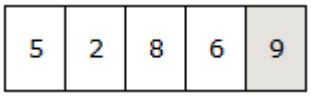
- Element 1 met element 2: 5<8 dus geen omwisseling.
- Element 2 met element 3: 8>2 dus deze getallen worden gewisseld.
  De tabel wordt nu:



- Element 3 met element 4: 8<9 dus geen omwisseling
- Element 4 met element 5: 9>6 dus deze getallen worden gewisseld.
  De tabel wordt nu:

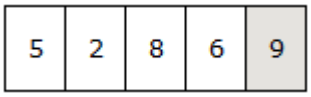


Het grootste element staat nu achteraan:



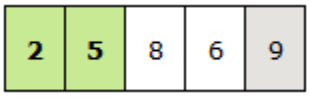
Tweede iteratie

Bij de tweede iteratie gaan we nog slechts tot het voorlaatste element.

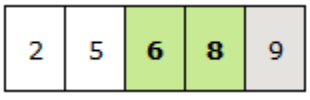


We vergelijken dus:

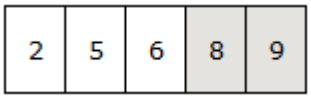
- Element 1 met element 2: 5>2 dus deze getallen worden gewisseld.
  De tabel wordt nu:



- Element 2 met element 3: 5<8 dus geen omwisseling.
- Element 3 met element 4: 8>6 dus deze getallen worden gewisseld.

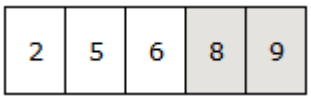


Het tweede-grootste element staat nu op de voorlaatste plaats:



Derde iteratie

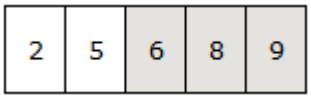
Bij de derde iteratie gaan we nog slechts tot het derde-laatste element.



We vergelijken dus:

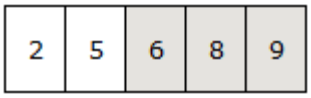
- Element 1 met element 2: 2<5 dus geen omwisseling
- Element 2 met element 3: 5<6 dus geen omwisseling

Het derde-grootste element staat nu op de derde-laatste plaats.



Vierde iteratie

Bij de vierde iteratie gaan we nog slechts tot het vierde-laatste element.



We vergelijken dus:

- Element 1 met element 2: 2<5 dus geen omwisseling

De tabel is nu gesorteerd.

2.3.3. Varianten

In het voorbeeld kon je al zien dat reeds bij de derde iteratie geen enkel element werd gewisseld. Wanneer je door de hele (op dat moment resterende) rij kan gaan ZONDER te wisselen, dan staat de rij reeds in de juiste volgorde.

Je kan dit implementeren door aan het begin van je iteratie een booleaanse variabele op false te zetten.
Telkens wanneer twee elementen worden gewisseld, zet je de variabele op true.
Indien aan het eind van je iteratie de variabele nog steeds op false staat, is er niets gewisseld. De rij staat al in de correcte volgorde en het sorteren mag stoppen.

Deze variant levert vooral serieuze tijdwinst op als je gegevens al grotendeels gesorteerd zijn, want dan worden een heleboel overbodige iteraties overgeslagen.

2.3.4. Opgave

Werk nu een procedure uit die **arrGetal** gaat sorteren via een Bubblesort.

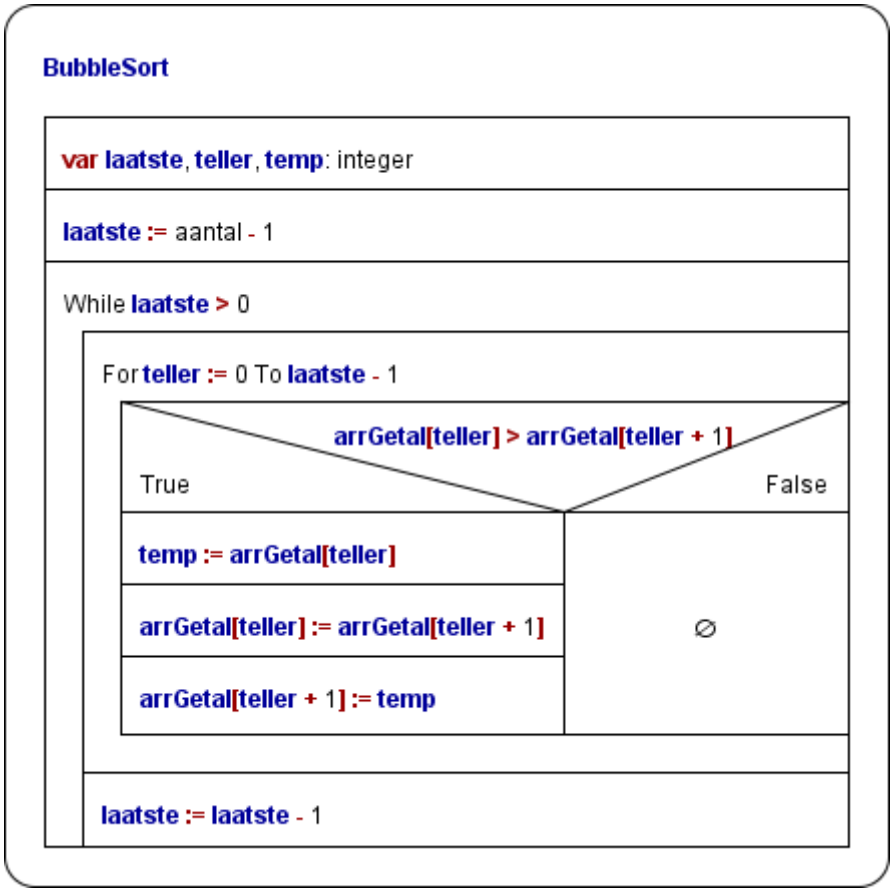
Algemeen geformuleerd ziet de werkwijze er zo uit:

- We onthouden de index van het laatste te sorteren element.

In het begin is dit dus het laatste element in de array. Daarna gaat die steeds afnemen tot hij gelijk is aan 1.

- Binnen elke iteratie (elke doorloop van de reeks) gaan we elk element met het volgende vergelijken. Als het element groter is als het element dat daarop volgt, worden ze omgewisseld.
- Dit vergelijken gebeurt maar tot het voorlaatste te sorteren element, omdat je het laatste element niet met een volgend element kan vergelijken.

Oplossing



2.4. Insertion sort

De **insertion sort** (of sortering door invoeging) is een iets efficiënter sorteeralgoritme. Dit is eigenlijk de manier waarop een speler zijn kaarten schikt bij een kaartspel. Vandaar dat we deze routine ook de **Cardsort** noemen.

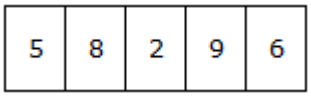
Er worden dezelfde vergelijkingen uitgevoerd als bij de bubblesort (zodat deze sortering even lang duurt), weliswaar in een andere volgorde:

- vertrek van een rij van 1 element (het eerste),
- plaats het tweede element op de juiste plaats in die rij (voor of achter het eerste),
- plaats het derde element op de juiste plaats t.o.v. de twee reeds gesorteerde elementen,
- ga zo verder tot het laatste element.
  Plaats dus telkens het N-de element op de juiste positie binnen de N-1 reeds gesorteerde

elementen.

2.4.1. Voorbeeld

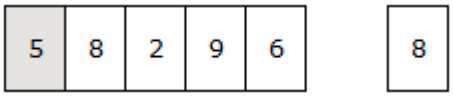
We vertrekken opnieuw vanuit dezelfde tabel:



Het eerste element wordt beschouwd als een reeds gesorteerde rij.

Eerste iteratie

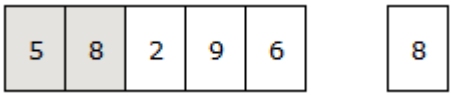
Bij de eerste iteratie gaan we het 2de element op de juiste plaats zetten. We onthouden de waarde van dat element even in een aparte locatie (we noemen dit hier **temp**, voor tijdelijke variabele).



Er worden dan de volgende vergelijkingen uitgevoerd:

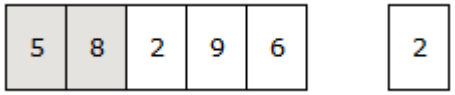
- **temp** met het 1ste element: **temp** >= 1ste (8 >= 5) dus er wijzigt niets.

Nu vormen de eerste twee elementen de *reeds gesorteerde rij*.



Tweede iteratie

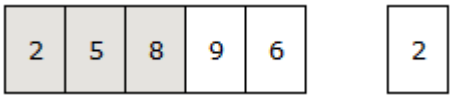
Bij de tweede iteratie gaan we het 3de element (2) op de juiste plaats zetten.



Er worden de volgende vergelijkingen uitgevoerd:

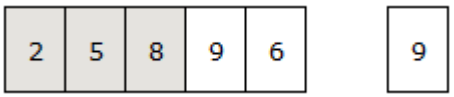
- **temp** met het 2de element: **temp** < 2de (2 < 8) en dus schuift het 2de element naar de 3de positie op.
- **temp** met het 1e element: **temp** < 1e (2 < 5) en dus schuift het 1e element naar de 2de positie op.
- **temp** wordt ingevoegd op de 1e positie.

Nu vormen de eerste drie elementen de *reeds gesorteerde rij*.



Derde iteratie

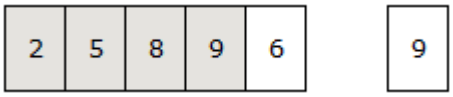
Bij de derde iteratie gaan we het 4de element (9) op de juiste plaats zetten.



Er worden de volgende vergelijkingen uitgevoerd:

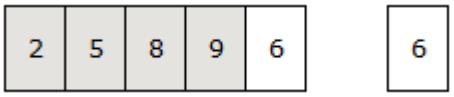
- **temp** met het 3de element: **temp** >= 3de (9 >= 8) dus er wijzigt niets.
- Vanaf nu zal er niets meer wijzigen vermits **temp** altijd >= zal zijn aan de elementen in de tabel. (**temp** is >= 3de element, dus ook >= 2e element en >= 1e element). Dat betekent dat je deze iteratie mag stoppen.

Nu vormen de eerste vier elementen de *reeds gesorteerde rij*.



Vierde iteratie

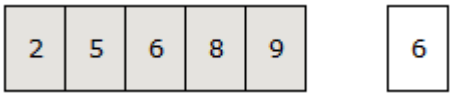
Bij de vierde iteratie gaan we het 5de element (6) op de juiste plaats zetten.



Er worden de volgende vergelijkingen uitgevoerd:

- **temp** met het 4de element: **temp** < 4de (6 < 9) en dus schuift het 4de element naar de 5de positie op.
- **temp** met het 3de element: **temp** < 3de (6 < 8) en dus schuift het 3de element naar de 4de positie op.
- **temp** met het 2de element: **temp** >= 2de (6 >= 5) dus er wijzigt niets.
- Vanaf nu is **temp** steeds >= alle elementen die nog volgen (hier: **temp** >= 1e).
  temp wordt ingevoegd op de 3de plaats (het laatste element dat verschoven werd) en de iteratie kan stoppen.

Nu is de hele reeks de *gesorteerde rij*.



2.4.2. Opgave

Werk nu een procedure uit die **arrGetal** gaat sorteren via een Insertion sort.

Algemeen geformuleerd ziet de werkwijze er zo uit:

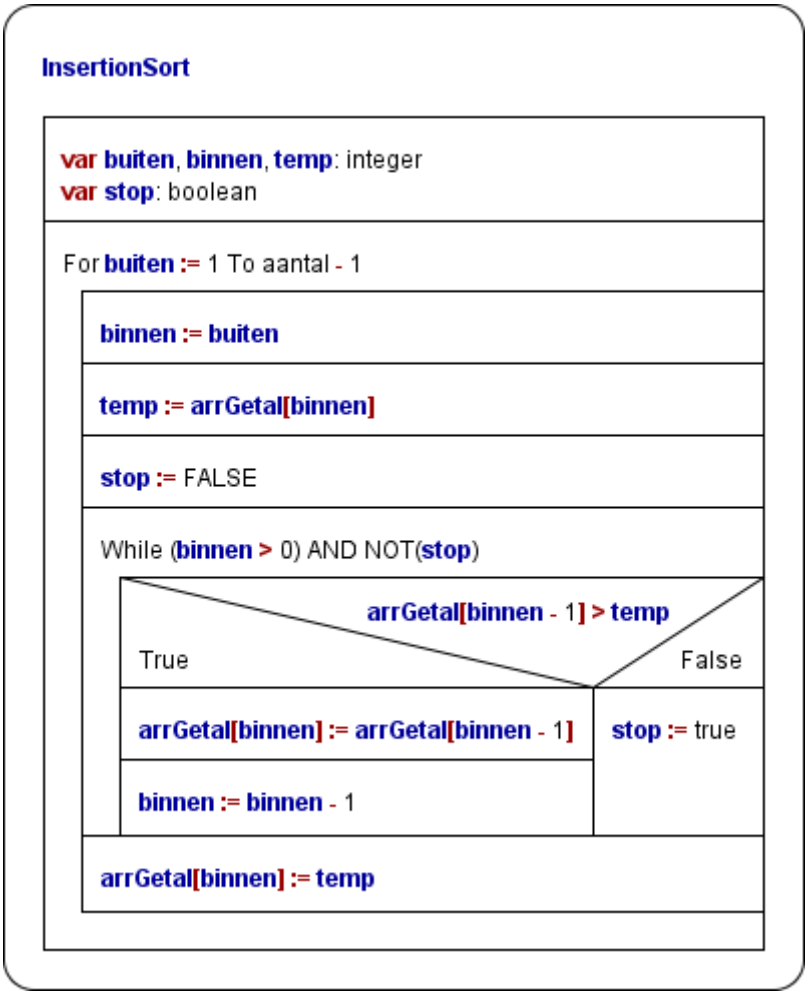
- Je houdt steeds de positie en de waarde bij van *het te plaatsen element*, dus het element dat je aan de reeds gesorteerde deelrij gaat toevoegen.

De grootte van de reeds gesorteerde deelrij is steeds één minder dan deze positie.

- Je vergelijkt het te plaatsen element met de elementen in de reeds gesorteerde deelrij in aflopende volgorde.
- Als de elementen groter zijn, schuiven ze een plaats op.
- Zodra je een element hebt gevonden dat kleiner is dan het te plaatsen element kan je stoppen met vergelijken en heb je de plaats van het te sorteren element gevonden.

Een dergelijke aftellende VOOR-lus is in Structorizer (net als in verschillende programmeertalen) niet mogelijk. We lossen dat op met een ZOLANG-constructie:

Oplossing



2.5. Quicksort

2.5.1. Principe

De quicksort is de meest complexe sorteermethode die we hier bespreken, maar ook de meest efficiënte. Samengevat komt het algoritme hierop neer:

- selecteer een spilelement (de pivot), meestal het middelste element
- zoek aan weerszijden een element dat daar “niet thuis hoort” (d.w.z. zoek rechts van de pivot een element dat kleiner is dan de pivot en links van de pivot één dat groter is dan de

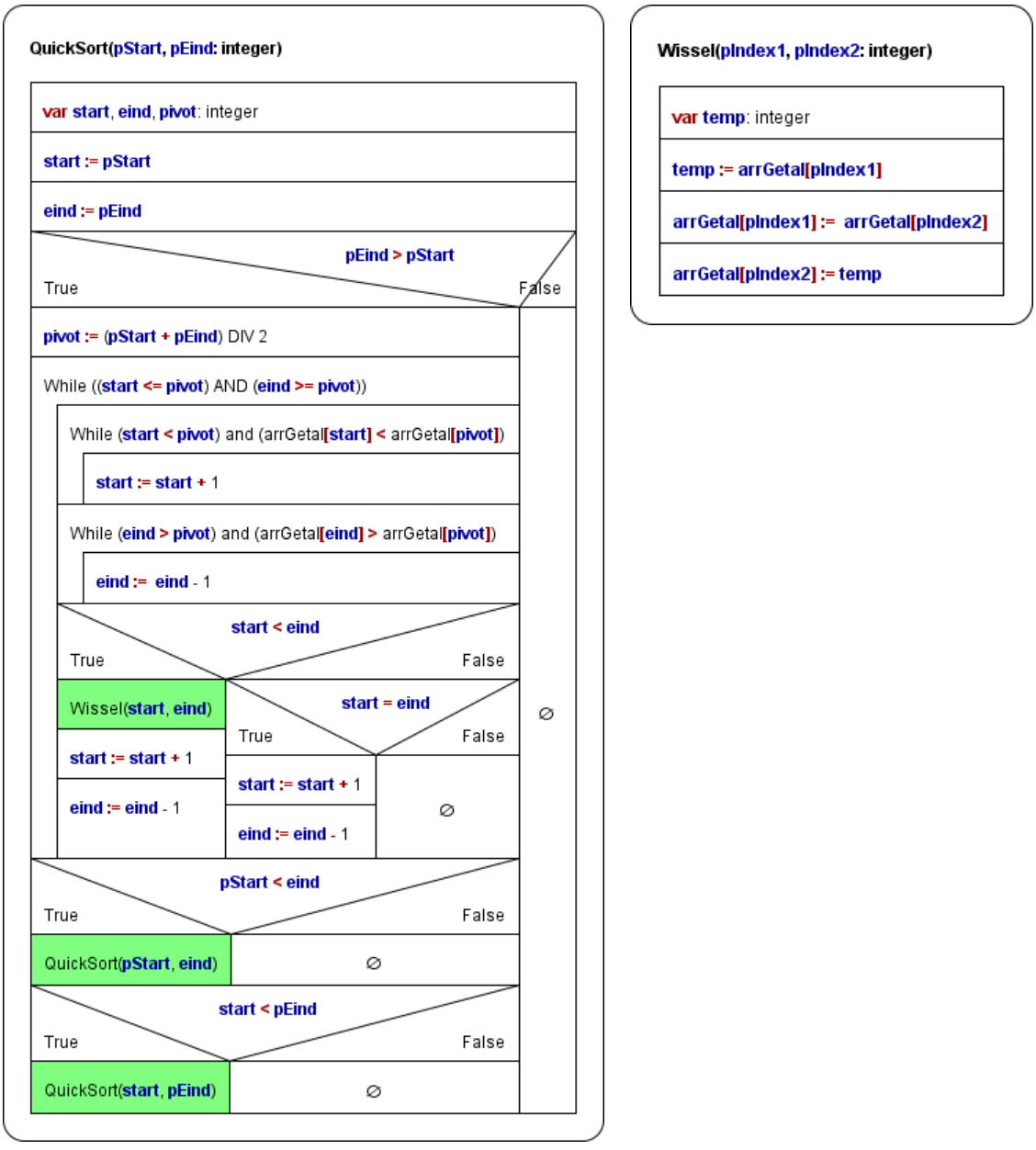
pivot)

- wissel die elementen
- zoek vanaf de gevonden posities verder in dezelfde richting, en doe telkens hetzelfde tot de twee “tellers” elkaar kruisen
- splits de reeks vervolgens op in twee deelreeksen (van links tot de positie van de rechterteller, en van rechts tot de positie van de linkerteller) en pas nu *recursief* ditzelfde algoritme toe op deze deelreeksen.

Je vindt hieronder het uitgewerkte PSD. Probeer zelf uit te vinden hoe dit precies in z’n werk gaat (bijvoorbeeld door een variabelentabel aan te maken).

Opgave

Maak deze psd na in Structorizer en test het uit in Lazarus.



2.6. Opgaven

In dit onderdeel maak je 5 oefeningen.

Vergeet niet de oplossing van de laatste oefening aan je **coach** te mailen ter verbetering!

2.6.1. Opgave 1: Selection sort - variant

Gegeven:
Vertrek van een tabel van 10 elementen, opgevuld met willekeurige getallen. (Je kan hiervoor je bestaande routine hergebruiken.)

Gevraagd:
Sorteer deze tabel van groot naar klein.

Principe:
Zoek telkens het kleinste element en plaats dit op de overeenkomstige plaats (te beginnen achteraan in de tabel).

Werkwijze:
Zoek het kleinste element in de tabel en verwissel het met het element op de laatste plaats. Zoek vervolgens het tweede kleinste element en verwissel het met het element op de voorlaatste plaats. Ga zo door tot de volledige tabel gesorteerd is.

2.6.2. Opgave 2: Bubblesort - variant

Maak een programma dat een lijst met (strikt positieve) cijfers inleest vanaf het toetsenbord en deze door middel van bubblesort sorteert. Er worden getallen ingegeven tot je een 0 ingeeft. Je mag maximaal 10 getallen inlezen.

Zorg ervoor dat je sortering stopt, zodra je weet dat de sortering correct is gebeurd.

2.6.3. Opgave 3: Karakters sorteren

Maak een programma dat een lijst met karakters inleest vanaf het toetsenbord, ingave van '*' stopt de invoer. Zorg dat er maximaal 10 tekens ingegeven kunnen worden. Kies zelf een sortering om deze lijst te sorteren.

Karakters worden volgens de ASCII-code met elkaar vergeleken. Let dus op: 'a' is groter dan 'Z'. (kleine letter a heeft als ASCII-waarde 97 en hoofdletter Z heeft als ascii-waarde 90).

Maak gebruik van de functies ord() om te zorgen dat je sortering niet hoofdletter gevoelig is.

2.6.4. Opgave 4: Quicksort

Maak een programma dat een lijst met (strikt positieve) cijfers inleest vanaf het toetsenbord. Er worden getallen ingegeven tot je een 0 ingeeft. Je mag maximaal 10 getallen inlezen. Sorteer deze getallen door middel van de quicksort.

2.6.5. Opdracht voor je coach: Namen sorteren

Maak een programma dat een aantal namen inleest in een tabel, en dat deze tabel achteraf alfabetisch sorteert (met een algoritme naar keuze).
Na het inlezen worden eerst de namen getoond, dus ongesorteerd, vervolgens worden de namen gesorteerd en daarna opnieuw getoond (in gesorteerde volgorde).

Let op: de sortering mag niet hoofdlettergevoelig zijn én de uitvoer mag ook op vlak van hoofdlettergebruik niet verschillen van je invoer.
Maak zinvol gebruik van procedures en/of functies!

Stuur je oplossing (zowel .nsd- als .pas-bestand) aan je **coach**. Vermeld ook **welk sorteeralgoritme** je gebruikt hebt.
Geef je bericht als onderwerp "**Namen sorteren**".

Hoofdstuk 3. Einde cursus

In dit laatste hoofdstuk overlopen we nog de volgende onderdelen:

- de eindoefening
- hoe je de afdrukbare versie van deze cursus kan downloaden
- wat je best doorneemt na deze module

3.1. Eindoefening

Bij deze cursus hoort ook een **eindoefening**. In die eindoefening komen de belangrijkste zaken van deze cursus terug aan bod.

Stuur een bericht aan je **coach** met als onderwerp "**Opdracht eindoefening Sorteren**".

3.2. Wat nu?

Je hebt nu het Programmatie logica – Sorteren afgerond.

Heb je het deel **Bestanden** nog niet doorgenomen, dan kan je dat nu doen.

Na het doornemen van **Basisstructuren**, **Procedures en functies** en **Arrays en strings** kan je voor de opleiding *Programmatielogica* een getuigschrift met resultaatsvermelding krijgen. Hiervoor leg je een eindtest af in één van de VDAB-opleidingscentra. De nodige informatie over de verschillende centra vind je terug in de module **Intro Informatica**.

Ligt je focus meer op **webapplicaties** en wil je graag al beginnen aan een concrete taal, dan zijn onze cursussen **Javascript** of **Inleiding tot PHP** misschien wel interessant voor jou.