2015

PSD

F. Vonk

versie 3

18-9-2015

inhoudsopgave

1. inleiding

Om software te kunnen maken moet je kunnen programmeren. Hiervoor is het belangrijk dat je snapt hoe je de structuur van een programma opzet en leest. De meeste programmeertalen zijn op tekst gebaseerd (tekstueel). Dat wil zeggen dat je in zo'n taal een programma intypt net als wanneer je een brief in bijvoorbeeld MS Word typt. Bij tekstuele programmeertalen is het moeilijk om de structuur van programma's, geschreven in zo'n taal, te doorgronden zonder veel ervaring met de taal te hebben. Daarom zijn er visuele hulpmiddelen (tools) ontworpen die het makkelijker maken de structuur van een programma te doorgronden en op te zetten. Met zo'n tool gaan we nu aan de slag. Dit tool heet Structorizer en maakt gebruikt van een techniek die Program Structure Diagram (PSD) heet.

Welkom bij de module *PSD*. We gaan in deze module uitleggen wat een PSD is en hoe je zo'n diagram leest en maakt in Structorizer.

In deze module kom je opgaves tegen die je moet maken om de lesstof te verwerken. De antwoorden kunnen in de les besproken worden.

> opgave
>
> Opgaves in blauw moet je maken.

Let op, links in dit document hebben een rode kleur.

Veel plezier en succes.

2. PSD

Zoals gezegd staat PSD voor Program Structure Diagram of Programma Structuur Diagram in het Nederlands wat toevallig dezelfde afkorting oplevert.

Figuur 1: voorbeeld van een PSD

In Figuur 1 zie je een voorbeeld van een PSD. Dit is een zogenaamd pseudo PSD, omdat je het niet uit kunt voeren. De instructies in dit PSD voldoen namelijk niet aan de regels die moeten gelden wanneer je een PSD wilt uitvoeren. Die regels ga je verderop stap voor stap leren. In het PSD uit Figuur 1 zie je de drie belangrijkste elementen van een programmastructuur. Deze drie elementen (ook wel programmeerconcepten genaamd) zijn:

- sequentie
- selectie (if)
- herhaling (lus of loop)

sequentie

Een sequentie is een serie stappen die één voor één en in volgorde worden doorlopen. Je kunt het vergelijken met een kookrecept, dit volg je typisch ook stap voor stap. In wezen is een programma niets anders dan een recept voor de computer.

Een stap uit een sequentie noemen we een statement, maar de term instructie wordt ook wel gebruikt.

Zoals je in Figuur 1 ziet kunnen sequenties op alle niveaus in de programmastructuur voorkomen. Je kunt dus sequenties op het hoogste niveau hebben maar ook in selecties en lussen.

selectie

selectie: kies `ja` pad als wat hier staat waar is kies `nee` pad anders ja	nee
in de `ja` sequentie stap A	in de `nee` sequentie stap I
in de `ja` sequentie stap B	in de `nee` sequentie stap II
	in de `nee` sequentie stap III

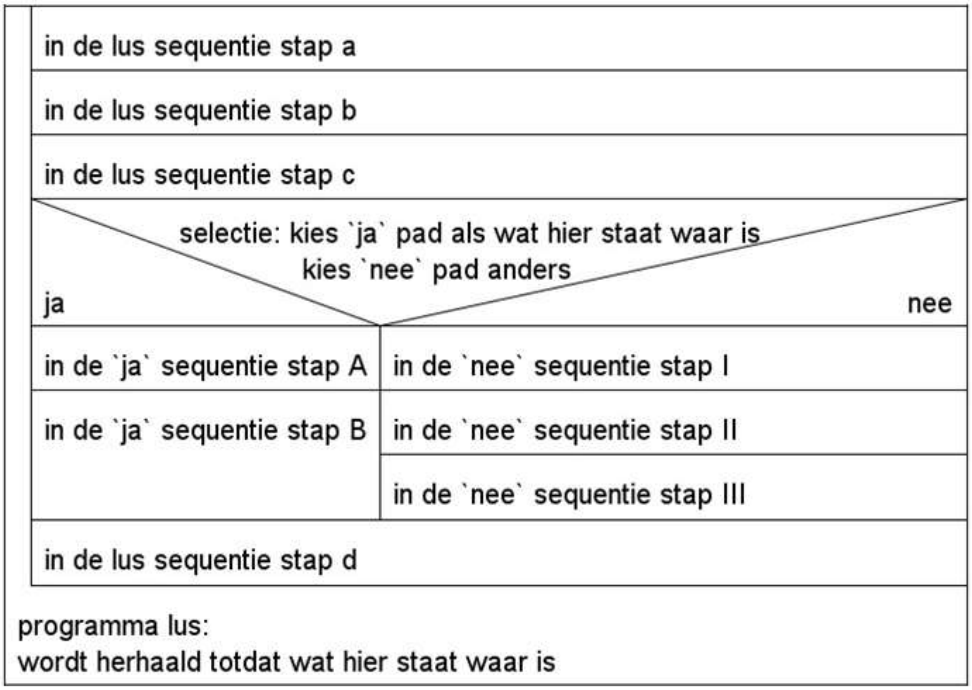
Figuur 2: De selectie uit Figuur 1.

Een selectie is een keuze moment. De computer moet hier kiezen welk pad van de code het gaat doorlopen. In Figuur 2 zie je twee paden maar dat kunnen er, in bijvoorbeeld een ander programma, ook meer zijn. De computer kan op het moment dat het een selectie in de code tegenkomt maximaal één van de paden doorlopen en niet meer.

Welk pad van de selectie de computer gaat nemen wordt bepaald door de conditie. Een conditie is een enkelvoudig of samengesteld predicaat (hopelijk weet je nog wat dat is uit de logica lessen) dat waar of onwaar is. Als de conditie waar is volgt de computer het "ja"-pad, anders (als de conditie dus onwaar is) volgt de computer het "nee"-pad.

In Figuur 1 staat de selectie in een lus maar net als sequenties kunnen selecties ook op alle niveaus voorkomen. Je kunt dus ook een selectie in een selectie hebben. Dit noemen we een geneste selectie.

herhaling



Een herhaling of lus (loop in het Engels) geeft aan dat de computer een stuk code een aantal keer moet herhalen. Als mens passen we ook wel eens herhaling toe. We voeren dan een aantal keer achter elkaar dezelfde activiteit uit. Stel je wilt op de fiets naar de stad en het regent, maar je weet via buienradar dat het zo droog kan worden. Je kijkt dan bijvoorbeeld iedere paar minuten even naar buiten om te controleren of het al droog is. Als je ziet dat het droog is dan pak je je jas en spullen en stapt op de fiets. Je hoeft vanaf dat moment niet meer steeds naar buiten te kijken om te controleren of het droog is want dat heeft geen zin meer, je zit immers al op de fiets. Met lussen kun je de computer iets soortgelijks laten doen.

In Figuur 1 zien we een lus zoals we die hiervoor geschetst hebben. De statements die bij de lus horen staan aan de rechterkant van de verticale balk die bij het PSD symbool voor de lus hoort. De verzameling van statements die bij een lus horen noemen we de lus-code.

opgave 2.1

Welke statements uit Figuur 1 horen bij de lus; oftewel behoren tot de lus-code?

Hopelijk heb je bij de opgave geantwoord dat het eerste statement van de lus `in de lus sequentie stap a` is en het laatste statement `in de lus sequentie stap d` is. Het statement `sequentie stap 4` is het eerste statement dat niet meer bij de lus-code hoort.

In de begindagen van het programmeren moesten programmeurs zelf hun lussen maken. Dit deden ze met behulp van springinstructies (jump of branch statements). In die dagen moesten regels code nog genummerd worden. Deze manier van programmeren is best illustratief voor hoe een lus werkt. Daarom laten we nu zien hoe het controleren of het al droog is (het aan het begin van deze paragraaf geschetste scenario) eruit zou zien als we dat moesten programmeren met behulp van springinstructies.

```
10 pak smartphone
20 start browser op
30 ga naar buienradar
40 wacht een paar minuten
50 kijk naar buiten
60 als het niet droog is, spring naar regel 40
70 pak jas en spullen
80 pak fiets
90 fiets naar stad
```

Het bovenstaande programma komt precies overeen met het scenario dat we beschreven hebben. Het gaat dus niet altijd van regel 60 naar regel 70 zoals dat bij de andere regels wel het geval is. Als het namelijk niet droog is springt het naar regel 40 en herhaalt het wachten en naar buiten kijken. Het gaat pas naar regel 70 als het wel droog is, want dan mag het niet naar regel 40 springen!

Hoe dit scenario eruit ziet in een PSD kun je zien in Figuur 3. Het PSD stopt met de lus als de conditie "het is droog" waar is. Dus als het niet droog is (conditie onwaar) springt het terug in het PSD, net zoals het programma hiervoor doet.

Figuur 3: PSD van beschreven scenario

Bij een lus gebruiken we dus ook een conditie, alleen gebruiken we die op een andere manier dan bij een selectie.

In Figuur 3 **herhaalt** het PSD het wachten en naar buiten kijken **totdat het droog is**. Daarom heet de lus die hier gebruikt wordt een totdat-lus of een repeat-until-loop. Een totdat-lus voert de lus-code dus uit totdat de conditie waar is. Op het moment dat de conditie waar is gaat de computer verder met het lezen van het PSD bij het eerste statement na de conditie.

opgave 2.2

Welk statement uit Figuur 1 en welk statement uit Figuur 3 wordt uitgevoerd meteen nadat de conditie van de lus waar wordt?

Hopelijk heb je bij de opgave voor Figuur 1 geantwoord dat dit *sequentie stap 4* is en voor Figuur 3 dat het *pak jas en spullen* is. Dit zijn namelijk de eerste stukken code die niet meer bij de lus-code horen.

Hoe vaak de computer een lus doorloopt wordt dus bepaald door de conditie. Hieronder zie je een mogelijke executie (uitvoering) van een totdat-lus door een computer:

1. voer lus-code uit
2. kijk of de conditie waar is
3. de conditie is onwaar, dus voer de lus-code uit
4. kijk of de conditie waar is
5. de conditie is onwaar, dus voer de lus-code uit
6. kijk of de conditie waar is
7. de conditie is waar, dus voer het eerste stuk code na de controle op de conditie uit

In deze executie wordt de lus-code drie maal doorlopen en de conditie wordt drie keer gecontroleerd. Bij een totdat-lus wordt de conditie altijd even vaak gecontroleerd als dat de lus-code wordt uitgevoerd. Dit is niet bij alle soorten lussen het geval zoals we later zullen zien. Bij stap 7 in de executie zeggen we dat de lus geeindigd is.

In Figuur 1 staat de lus op het hoogste niveau in de programmastructuur. Maar net als sequenties en selecties kunnen lussen op alle niveaus voorkomen. Je kunt dus ook een lus in een selectie hebben maar ook een lus in een lus. Dat laatste noemen we een geneste lus.

Misschien kun je je nog herinneren dat we in de module *talstelsels* al een PSD hebben laten zien. In Figuur 4 zie je dit PSD nog een keer.

opgave 2.3

Schrijf in gewone tekst op wat er in het PSD uit Figuur 4 gebeurt en beantwoord de volgende vragen:

a) Is het statement `Neem een decimaal getal` onderdeel van de lus-code?
b) Is het statement `Deel het decimale getal door 2` onderdeel van de lus-code?
c) Is het statement `Trek van het decimale getal 1 af` onderdeel van de lus-code?
d) Is het statement `Ga door tot het decimale getal gelijk is aan 0` onderdeel van de lus-code?

Figuur 4: PSD voor omrekenen decimaal naar binair

opgave 2.4

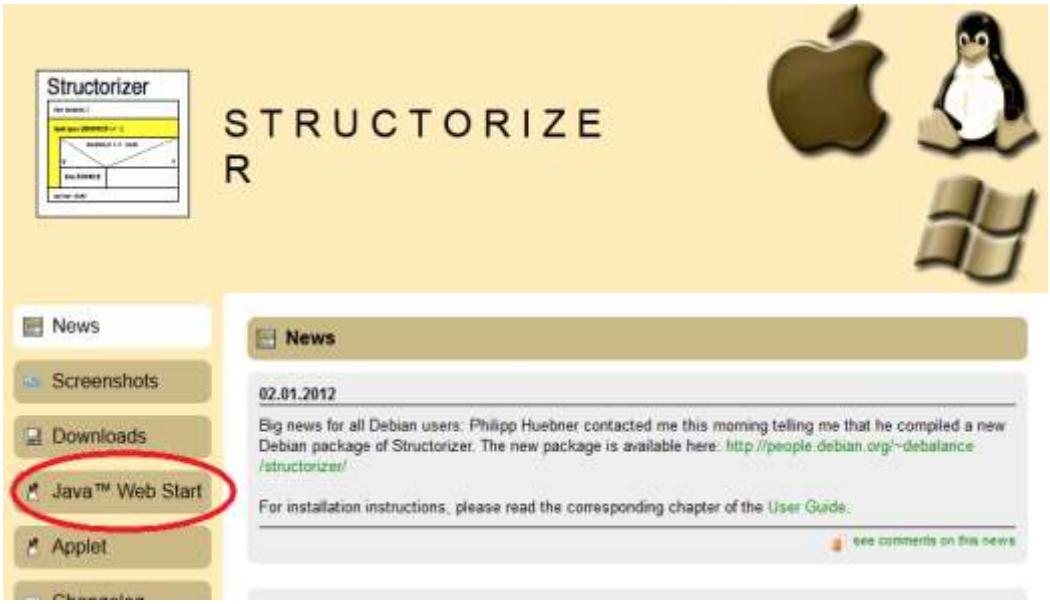
Waar of niet waar?

a) Geneste sequenties bestaan.
b) Een sequentie bestaat uit statements ook wel instructies genaamd.
c) Sequenties komen alleen op het hoogste niveau in de programmastructuur voor.
d) Bij een selectie doorloopt de computer alle mogelijke paden van de selectie.
e) Bij een selectie kiest de computer op basis van de conditie welke van de paden als eerste doorlopen wordt.
f) Een selectie kan in een andere selectie staan en zelfs ook in een lus.
g) Een herhaling (of lus) bestaat uit een conditie en lus-code.
h) De lus-code kan in principe oneindig vaak uitgevoerd worden.
i) Als bij een totdat-lus de conditie onwaar wordt dan eindigt de lus.
j) Als een totdat-lus eindigt dan gaat de computer verder bij het statement na de controle op de conditie.
k) Bij een totdat-lus wordt de lus-code even vaak uitgevoerd als dat de conditie wordt gecontroleerd.
l) Een lus kan in een andere lus staan maar niet in een selectie.

3. Structorizer

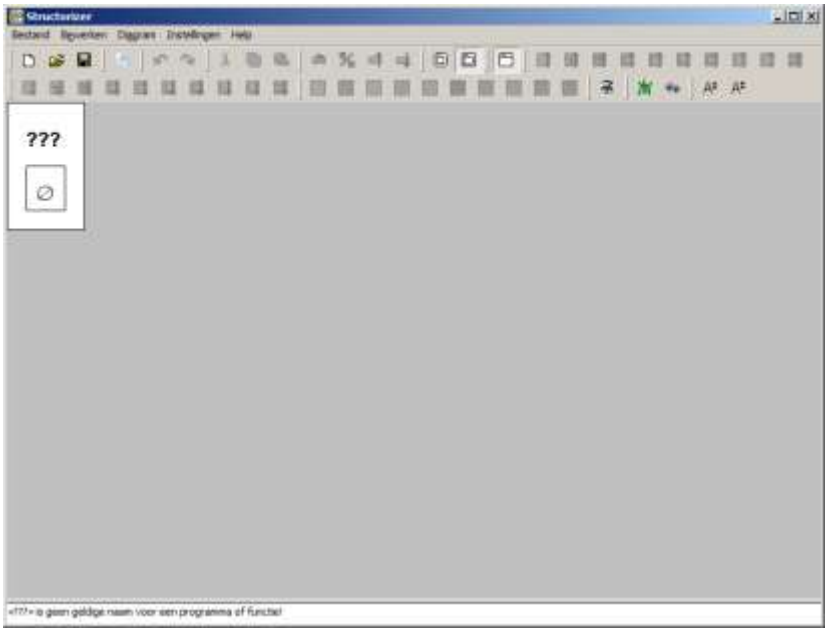
Even genoeg over programmeerconcepten. Laten we eens kijken hoe ons tool werkt.

We kunnen Structorizer online gebruiken. Dit doen we via de structorizer.fisch.lu website. Nadat je naar de website bent gegaan, klik je op Java Web Start zoals aangegeven is in Figuur 5. Het kan zijn dat je een aantal keer moet bevestigen dat je het programma daadwerkelijk wilt uitvoeren.



Figuur 5: Java Web Start link op site

Nadat je de online versie hebt opgestart, zie je als het goed is wat in Figuur 6 is afgebeeld. Feitelijk heb je nu je webbrowser niet meer nodig, dus die kun je afsluiten als je wilt.

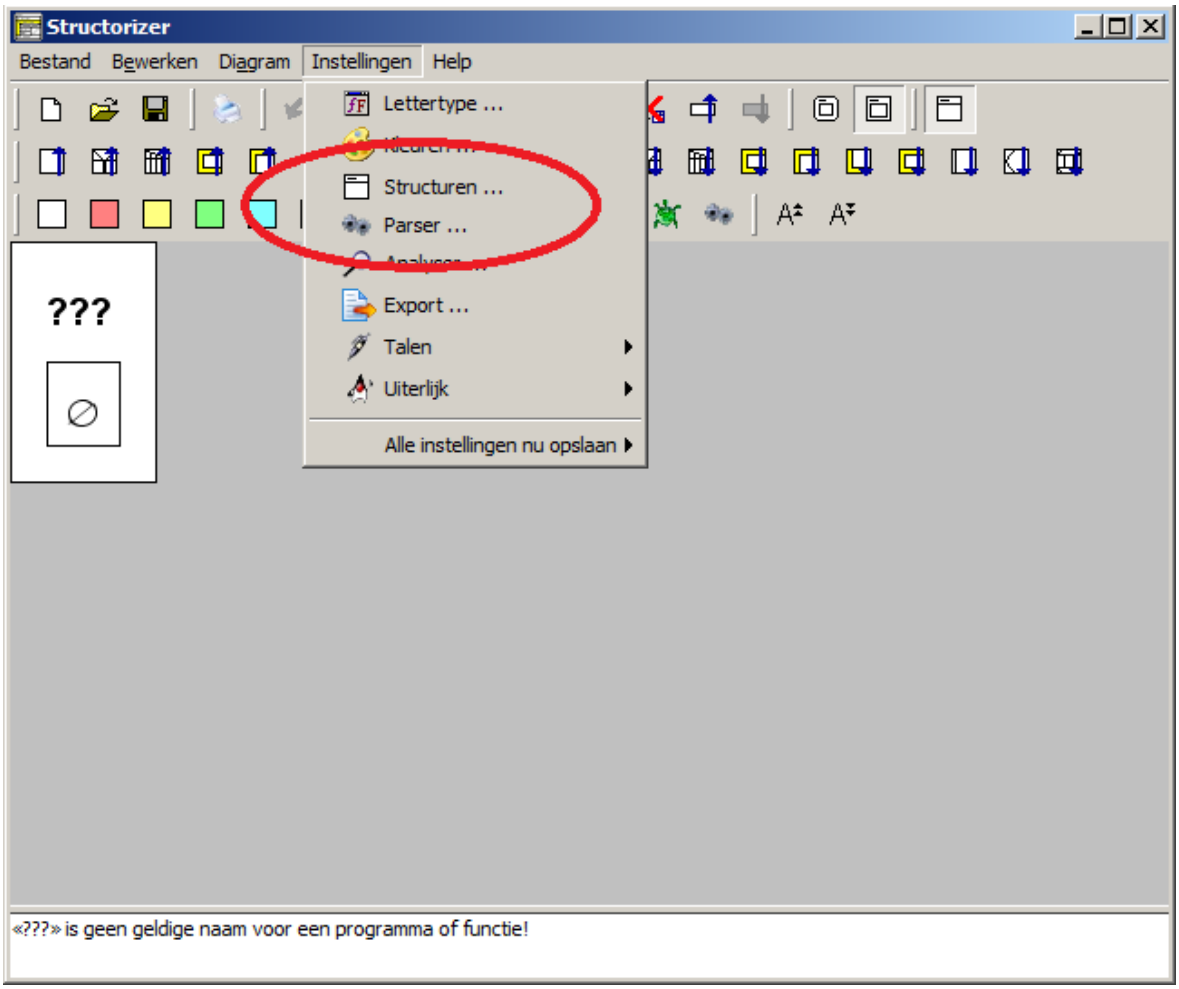


Figuur 6: Structorizer online

Als je op een andere PC zit dan waar je normaal zit, of als je Structorizer voor het eerst gebruikt, dan moet je het programma eerst configureren zoals hierna beschreven is. Het kan echter geen kwaad om altijd even te controleren of het programma goed geconfigureerd is.

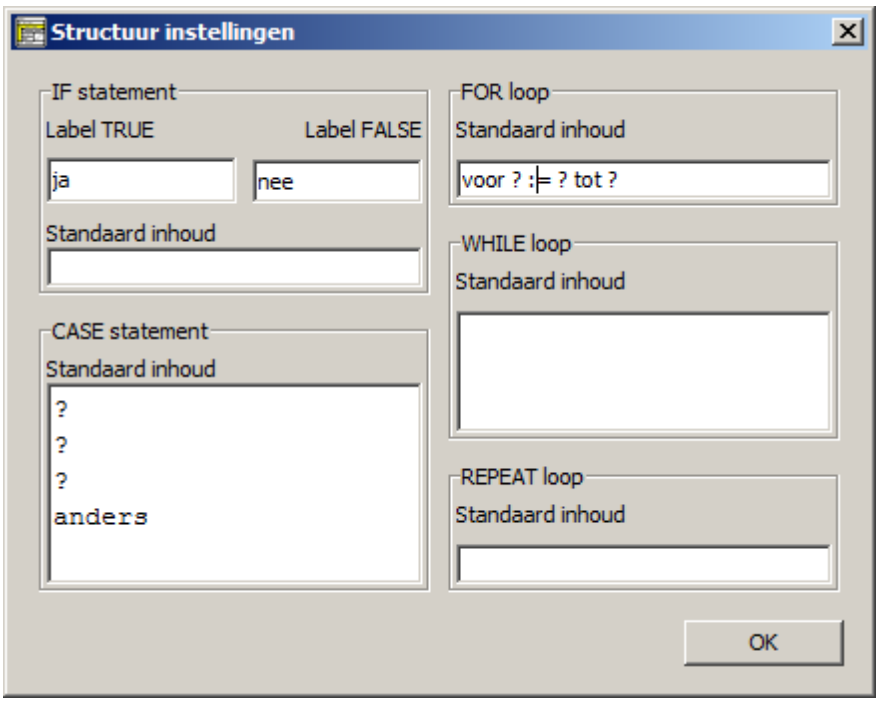
configuratie

Kijk onder *Instellingen* bij zowel *Structuren* als *Parser*, zie Figuur 7.

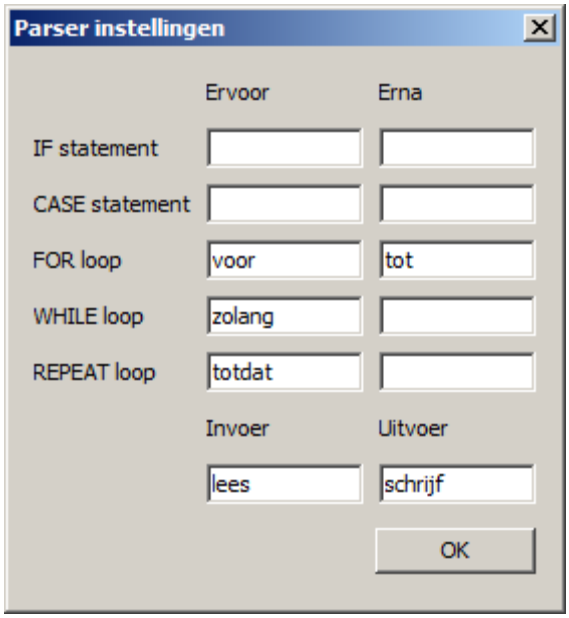


Figuur 7: Structorizer instellingen

De aangeraden instellingen voor *Structuren* staan in Figuur 8. De aangeraden instellingen voor Parser staan in Figuur 9.



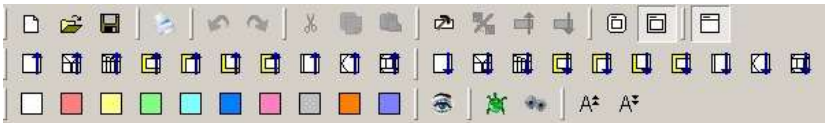
Figuur 8: Structorizer Structuur instellingen



Figuur 9: Structorizer Parser instellingen

gebruik

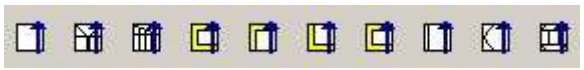
We gaan nog geen echte programma's schrijven maar eerst wennen aan de interface van Structorizer. In Hoofdstuk 2 heb je een drietal programmeerconcepten gezien, namelijk de sequentie, de selectie en de lus. Deze gaan we straks toepassen. Laten we eerst eens naar het instrumentpaneel van Structorizer kijken. Dit is afgebeeld in Figuur 10.



Figuur 10: Structorizer instrumentenpaneel

Een aantal pictogrammen komt je waarschijnlijk bekend voor, zoals die voor "nieuw diagram", "open diagram" en "sla diagram op". Deze staan links bovenin. Als je het programma overigens full screen draait dan is het instrumentenpaneel breder en staan de pictogrammen anders.

In Figuur 11 en Figuur 12 zien we de middelste rij uit Figuur 10 uitgelicht. Deze rij bevat namelijk de instrumenten die belangrijk zijn om structuurdiagrammen te tekenen. De instrumenten uit Figuur 11 en Figuur 12 tekenen dezelfde structuurelementen alleen voegen ze die op een andere plaats in het bestaande diagram in. De richting van de pijlen in de pictogrammen geeft aan of een element boven of onder het geselecteerde element wordt ingevoegd. De volgorde van instrumenten is in beide figuren hetzelfde.



Figuur 11: invoegen boven geselecteerd element

Figuur 12: invoegen onder geselecteerd element

Het eerste instrument is de sequentie. Het tweede instrument is de selectie met twee paden. Dit is de selectie die je het meest zult gebruiken. Het derde element is de selectie met meer dan twee paden. Daarna komen vier verschillende soorten lussen. De laatste drie instrumenten hebben we niet nodig. De lus die we hiervoor hebben gezien, de totdat-lus, is het zesde instrument.

We gaan instrumenten 1, 2 en 6 nu gebruiken om te wennen aan het tool.

Zoals we in Figuur 6 hebben gezien beginnen we met een leeg diagram. In de bovenste regel staan drie vraagtekens. Dit is de ruimte voor de naam van het PSD.

opgave 3.1

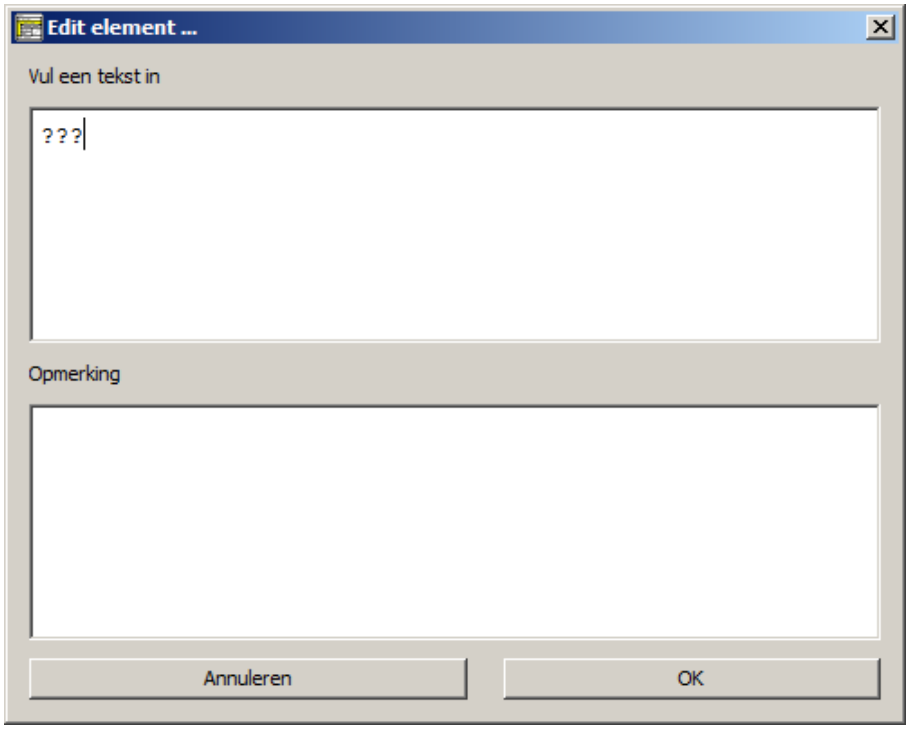
Doorloop de volgende stappen:

1. Selecteer de vraagtekens. Dit doe je door erop te klikken. Gebruik een enkele klik. Als je dubbelklikt terwijl een element nog niet geselecteerd is, dan gedraagt Structorizer zich vreemd.
2. Dubbelklik op de vraagtekens.
   Als het goed is zie je nu de pop-up uit Figuur 13.
3. Vul het volgende in als naam: van decimaal naar binair
   Vergeet niet de vraagtekens weg te halen.
4. Klik op OK.
5. Selecteer het vlak waarin het symbool voor leeg (Ø) staat.

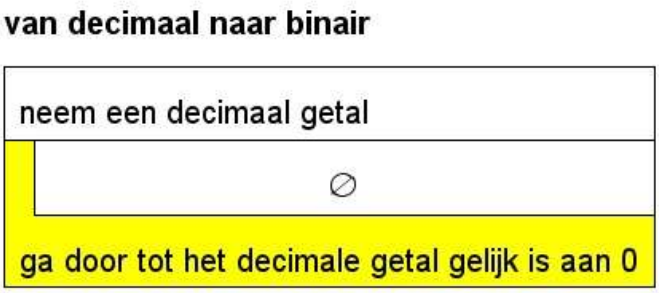
6. Dubbelklik op de selectie en voer het eerste statement (instructie) in te weten: neem een decimaal getal
7. Klik op OK. Als het goed is blijft het zojuist veranderde element geselecteerd. Het geselecteerde element heeft de kleur geel terwijl niet geselecteerd element wit zijn.
8. Voeg nu een totdat-lus toe onder het geselecteerde element. Hiervoor gebruik je dus het zesde instrument uit Figuur 12.
   Als het goed is zie je nu de pop-up uit Figuur 15.
9. Vul de conditie van de lus in te weten: ga door tot het decimale getal gelijk is aan 0
10. Klik op OK.
11. Als het goed is heb je nu in de lus een vlak gekregen waarin het symbool voor leeg (Ø) staat.

Hopelijk zie je nu wat in Figuur 14 is afgebeeld. Maak dit PSD nu af aan de hand van Figuur 4. Vergeet niet elementen te selecteren voor je ze dubbelklikt. Je hebt alleen sequenties en een selectie nodig. **Sla dit PSD op!**

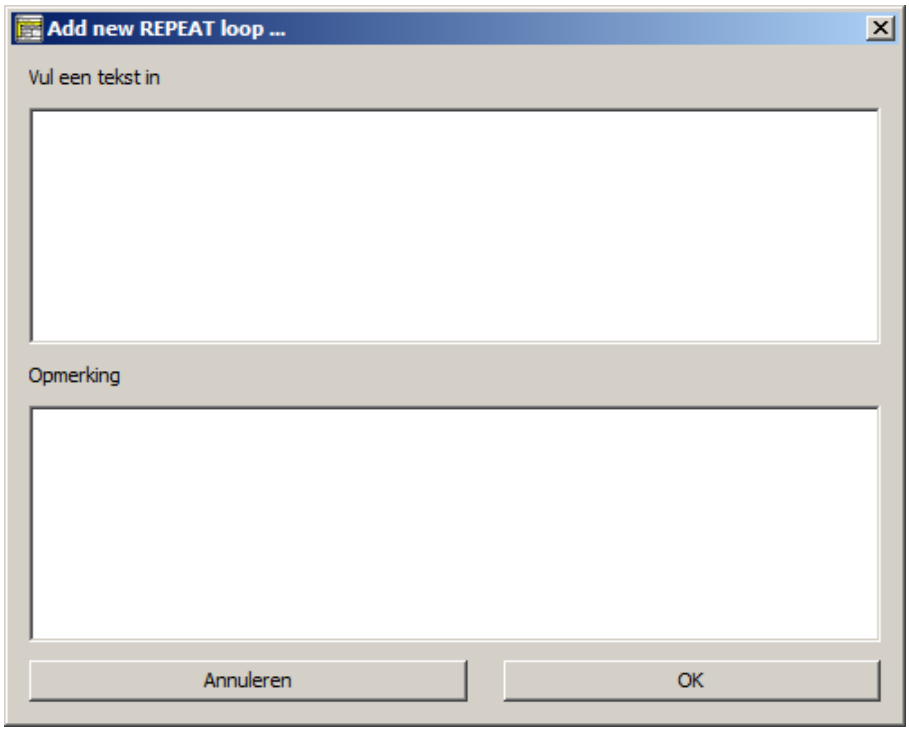
Let verder eens op wat er helemaal onderin je Structorizer scherm staat. Hier zie je de foutmeldingen. Zo zie je bijvoorbeeld dat Structorizer wil dat de programmanaam in hoofdletters geschreven is.



Figuur 13: Structorizer Edit element pop-up



Figuur 14: het PSD uit opgave 3.1 tot zover



Figuur 15: Structorizer voeg totdat-lus toe pop-up

Je hebt nu wat gespeeld met Structorizer en oefening baart kunst. Daarom komen hier nog een paar opdrachten.

opgave 3.2

Maak het PSD uit Figuur 1 na.

Als je naar de foutmeldingen bij het laatst gemaakte PSD kijkt, dan zie je dat je nog niet echt aan het programmeren bent. Als je de computer vraagt dit PSD uit te voeren dan lukt dit niet omdat de computer het PSD niet snapt. Computers zijn nu eenmaal niet zo goed in het lezen van Nederlandse tekst.

Het mooie aan Structorizer is dat het de diagrammen die je hebt gemaakt ook echt kan uitvoeren, mits deze aan de regels voldoen. Misschien herinner je je nog dat je tijdens het configureren onder andere woorden als "lees" en "schrijf" hebt opgegeven. Met "lees" geven we Structorizer nu de opdracht om invoer van het toetsenbord te lezen en met "schrijf" om uitvoer naar het beeldscherm te sturen.

Hopelijk heb je het PSD om decimale getallen naar binaire getallen te converteren opgeslagen. We gaan nu proberen dit om te bouwen zodat de computer het kan uitvoeren.

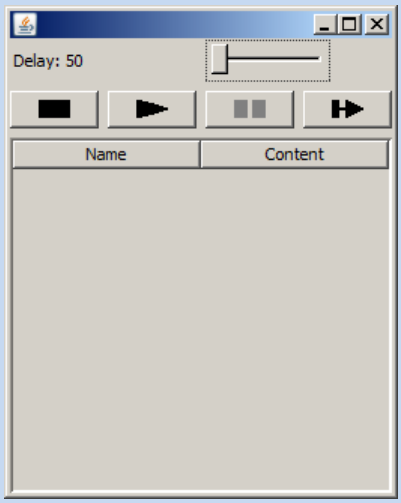
opgave 3.3

Doorloop de volgende stappen:

1. Verander de naam van het PSD in: DEC2BIN
2. Verander de instructie "neem een decimaal getal" in: lees getal
   Als het goed is wordt het woord "lees" nu groen. Dat wil zeggen dat Structorizer het herkent.
3. Verander de conditie van de selectie in: getal % 2 = 0.
   Als het goed is wordt het = symbool nu rood. Dat wil zeggen dat Structorizer het herkent. Maak je nog maar niet druk om het % symbool. Daar wordt verderop in het curriculum uitgelegd.
4. Verander de instructie in het "ja" pad in: schrijf 0
5. Verander de bovenste instructie in het "nee" pad in: schrijf 1
6. Verander de onderste instructie in het "nee" pad in:
   getal := getal - 1
7. Verander de instructie "deel het decimale getal door 2" in:
   getal := getal / 2.
8. Verander de conditie van de lus in: getal = 0

Nu ben je klaar om het PSD uit te voeren. Hiervoor gebruik je het "run program" instrument van Structorizer. Dit is het op twee na laatste instrument in het instrumentenpaneel en ziet er als volgt uit.

Als je hier op drukt krijg je de volgende pop-up.



Met de play knop (tweede van links) kun je het PSD uitvoeren. Als je het uitvoert en je schrijft de cijfers, die je in de pop-ups ziet, van rechts naar links op dan heb je het goede antwoord. De uitvoer is ietwat raar maar voorlopig goed genoeg.

Met de meest rechtse knop (de step knop) kunnen we stap voor stap door het PSD lopen. Dit kun je gebruiken om te debuggen, want door te steppen kun je vaak makkelijker fouten ontdekken dan door naar het PSD zelf te kijken.

Als je het uitvoeren van het PSD wilt verlaten druk je op de meest linkse knop (de stop knop).

Zo, nu heb je een PSD daadwerkelijk uit laten voeren door de computer. Is het je overigens opgevallen dat Structorizer nog steeds foutmeldingen geeft ondanks dat het PSD goed uitgevoerd wordt. Structorizer heeft helaas geen geweldige controle op wat er precies fout gaat, het zijn meer hints die je krijgt om je PSD te verbeteren.

Het PSD om decimale getallen om te rekenen naar binaire is niet echt moeilijk maar ook niet heel makkelijk om mee te beginnen. Om zelf te oefenen beginnen we met eenvoudigere problemen. Bijvoorbeeld het maken van een PSD dat bepaalt of iemand wel of niet bij moet betalen omdat zijn koffer te zwaar is bij een vliegreis. Laten we zeggen dat iemand bij moet betalen als zijn/haar koffer zwaarder is dan 20 kilogram. Hoe laten we een computer dit bepalen?

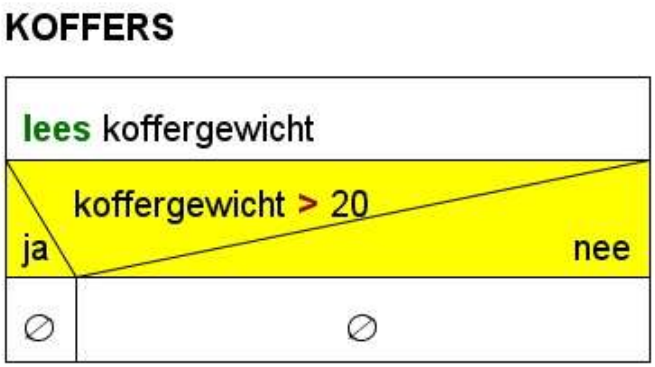
Allereerst moeten we het PSD vertellen hoe zwaar de koffer is. We moeten dus gebruikersinvoer kunnen lezen en dat doen we met "lees". Het complete statement ziet er als volgt uit: `lees koffergewicht`.

Zoals we eerder hebben gezien krijgen we hierdoor een pop-up die ons vraagt iets in te typen. In het geval van een gewicht zullen we typisch een getal intypen. Dit getal wordt vervolgens door het PSD bewaard en wel in iets dat `koffergewicht` heet. We noemen `koffergewicht` een variabele. Een variabele is dus als het ware een doos waar het PSD iets in kan stoppen en bewaren.

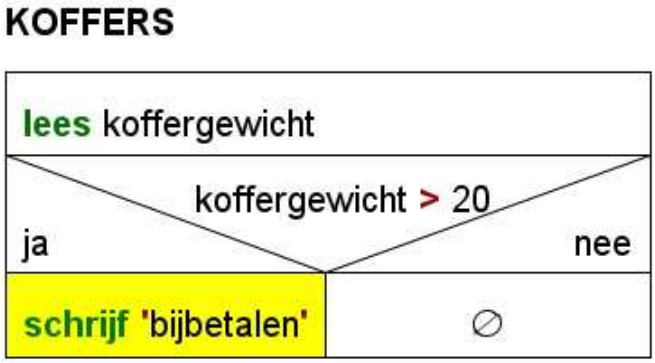
> Een variabele is een verwijzing naar een geheugenlocatie waarin data staat. Via een variabele kun je gegevens in het geheugen schrijven en er ook weer uit lezen.
>
> Variabelen zijn nodig om de toestand van een programma te onthouden. In variabelen sla je data op die later nodig is en die je dan niet opnieuw wilt of kunt opzoeken of berekenen.

Als we onmiddellijk na het `lees koffergewicht` statement het statement `schrijf koffergewicht` toevoegen en het PSD uitvoeren, dan krijgen we een pop-up die ons vertelt welk getal we zojuist ingevoerd hebben. In het eerste statement wordt de variabele geschreven, in het tweede gelezen. Probeer het maar eens.

Als we het gewicht van de koffer aan het PSD verteld hebben moeten we het PSD een beslissing laten nemen. Er zijn twee mogelijke uitkomsten en daarom hebben we een selectie met twee paden nodig. In de conditie moeten we bepalen of het gewicht meer dan 20 kilogram is of niet. Dat kunnen we uitdrukken door middel van de conditie `koffergewicht > 20`.

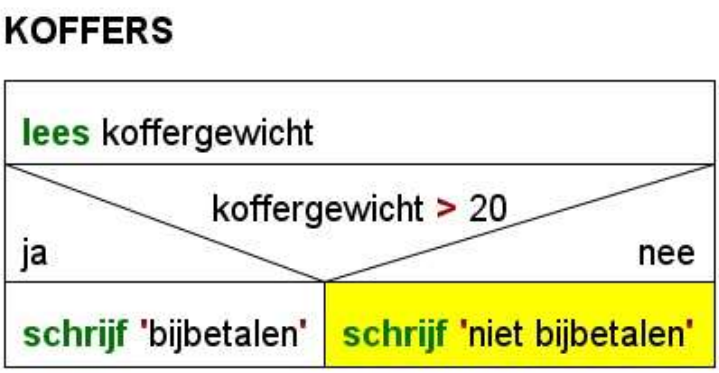


In het "ja"-pad is er dus sprake van een koffer van meer dan 20 kilogram en moet er bijbetaald worden. Om dit aan te geven kunnen we bijvoorbeeld het statement `schrijf 'bijbetalen'` gebruiken.



Structorizer kent het woord *bijbetalen* niet. We moeten het PSD daarom vertellen dat het dit woord niet moet proberen te interpreteren maar alleen af moet afdrukken. Hiervoor kunnen we een enkele quote (') gebruiken. Als we zo'n quote voor en achter een stuk tekst zetten, dan zal Structorizer negeren wat er tussen staat en dit simpelweg afdrukken. We noemen het stuk tekst met een quote ervoor en erna een string. In plaats van een enkele quote mag je ook een dubbele quote (") gebruiken.

In het "nee"-pad hoeven we niet bij te betalen. Hopelijk kun je daar nu zelf een statement voor bedenken.



En dan is het PSD klaar. Simpel toch, of niet? Je kunt het PSD nabouwen, uitvoeren en testen of het werkt zoals het hoort.

opgave 3.4

Maak een PSD dat afdrukt of iemand meer- of minderjarig is.

opgave 3.5

Maak een PSD dat afdrukt of het voor of na de middag (12 uur) is.

Misschien begrijp je nog niet alles wat je hiervoor hebt gebruikt. We hebben bijvoorbeeld nog niet uitgelegd wat de symbolen % en `:=` doen. Ook begrijp je sommige statements zoals `getal := getal / 2` misschien nog niet helemaal. Toch ben je nu al aan het programmeren en dat is een mooi begin.

opgave 3.6

Waar of niet waar?

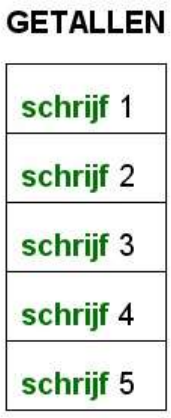
a) Als Structorizer foutmeldingen geeft in het scherm onderin dan betekent dit dat je PSD het niet gaat doen.
b) Een variabele is een soort doos waarin een PSD dingen op kan slaan zoals getallen.
c) Variabelen kun je alleen schrijven maar niet lezen.
d) In condities kunnen we vergelijkingsoperatoren gebruiken zoals <, <=, > en >=.

In het volgende hoofdstuk gaan we nog iets dieper als je zelf met lussen gaat oefenen.

4. de totdat lus

Herhaling (lus of loop) is één van de moeilijkere programmeerconcepten. Waarschijnlijk omdat het niet helemaal intuïtief is. Daarom is het belangrijk er veel mee te oefenen.

Stel je wilt de getallen 1 tot en met 5 door de computer laten genereren via een PSD. "Simpel", zeg je dan en maakt het PSD dat je ziet in Figuur 16.



Figuur 16: een reeks getallen

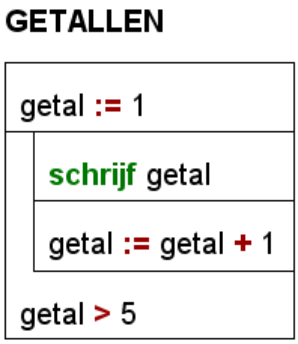
opgave 4.1

Maak het PSD uit Figuur 16 na.

Hopelijk vond je het saai worden vanaf getal 4. Dat is namelijk een goed teken, want als het saai begint te worden betekent dit dat je iets niet optimaal doet. Stel dat je niet de getallen 1 tot en met 5 wilt laten genereren maar de getallen 1 tot en met 500. Zou je dit dan op dezelfde manier oplossen? Hopelijk niet! De easy way out zou het PSD uit Figuur 17 kunnen zijn maar dat kan een computer niet uitvoeren.

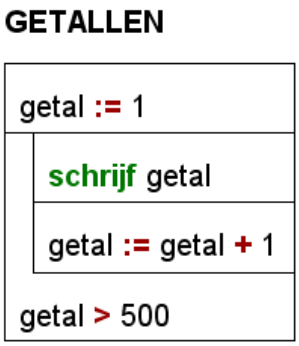
Figuur 17: nog een reeks getallen

Bij het programmeren zijn lussen vaak een oplossing als het saai begint te worden. We kunnen het eerste probleem namelijk ook oplossen zoals in Figuur 18 is gedaan.



Figuur 18: een reeks getallen via een lus

Je zult misschien zeggen: "Dat scheelt maar één statement, wat zijn we daar nu mee opgeschoten?" Nou, in het eerste geval moesten we feitelijk vijf keer hetzelfde statement intypen (op het cijfer zelf na dan). Met de lus oplossing hebben we vier totaal verschillende statements en dat is al minder saai. Er is echter nog een veel belangrijkere reden en dat is dat de oplossing met de lus opschaalt. Daarmee bedoelen we dat als we meer getallen willen genereren we nauwelijks meer statements nodig hebben en de bestaande structuur nauwelijks hoeven aan te passen. Als we namelijk de getallen 1 tot en met 500 willen laten genereren dan ziet ons PSD eruit zoals Figuur 19. Als je goed kijkt zie je dat we maar één getal hebben aangepast. We hebben de 5 in 500 veranderd en dat was het. En ineens hebben niet 5 maar 500 getallen. Makkelijk of niet?



Figuur 19: nog een reeks getallen via een lus

opgave 4.2

Maak een PSD dat de getallen 1 tot en met 1000 genereert.

HINT: Maak eerst het PSD dat de getallen 1 tot en met 5 genereert, test of het werkt en maak daarna pas het uiteindelijke PSD. Het is beter om het uiteindelijke PSD niet uit te voeren aangezien je dan nogal wat moet klikken of op enter moet drukken voor je klaar bent.

Ongetwijfeld vraag je je nu af hoe dit allemaal precies werkt. Hopelijk herinner je je de uitleg van de totdat-lus nog. Zo niet ga dan terug naar Hoofdstuk 2 en zoek het op, want het is belangrijk om het lus concept te snappen.

Als je weer snapt hoe de totdat-lus precies werkt, dan kunnen we een aantal nieuwe aspecten van lussen gaan uitleggen.

Weet je nog dat we naar de stad wilden en dat het regende? Stel je nu eens voor dat het nooit op zou houden met regenen. Hoe lang zouden we dan naar buiten blijven kijken als we simpelweg ons "programma" zouden volgen? Precies, ons hele leve lang. We zouden er nooit meer mee ophouden. Gelukkig zijn mensen flexibel en wijken we af van ons "programma" als het te zot wordt. Computers hebben die luxe echter niet. Een computer zal altijd zijn voorgeschreven programma blijven volgen al duurt het oneindig lang. Bij lussen is dit gevaar voor computers heel reeel. Als namelijk de conditie van onze totdat-lus nooit waar wordt dan zal de computer oneindig lang de lus uitvoeren, we noemen dit een oneindige lus. Soms willen we dat een programma een oneindige lus uitvoert maar meestal gaat dat per ongeluk en is het ongewenst.

opgave 4.3

Waar zit het Structorizer instrument voor de oneindige lus?

Hiermee hebben we een nieuw aspect van lussen aangesneden, namelijk het eindigen van lussen.

Als je wilt dat een totdat-lus eindigt, dan moet de conditie van de lus ooit waar worden.

Dit is een heel belangrijke regel. Maar hoe krijgen we dat voor elkaar? Als je naar Figuur 19 kijkt dan zie je dat er een variabele in de conditie staat. Het mooie aan variabelen is dat we ze zo vaak een nieuwe waarde kunnen geven als we willen. We zeggen dan dat de variabele van waarde verandert. We kunnen de conditie van de lus dus waar maken door de variabele, die in de conditie gebruikt wordt, op de juist manier te veranderen.

Als we een variabele willen veranderen dan is het belangrijk dat de variabele een waarde heeft. Als dat namelijk niet zo is dan weten we niet altijd wat voor effect onze verandering op de variabele heeft. Dit kun je vergelijken met het plannen van een dagtrip naar bijvoorbeeld De Efteling. Als je niet weet van waar je start kun je ook geen route naar De Efteling plannen. Hetzelfde geldt voor variabelen, je moet ze een startwaarde geven. Dit noemen we het initialiseren (begin waarde geven) van een variabele. In Figuur 19 is het eerste statement de initialisatie. We geven hier de waarde 1 aan de variabele `getal`. En dit doen we door het `:=` symbool te gebruiken. Dit symbool noemen we het toekenningssymbool en het hele statement heet een toekenning. Als we deze toekenning moeten verwoorden dan schrijven we: "De variabele `getal` wordt 1." Het woord "wordt" is belangrijk omdat het verandering aangeeft. In dit geval veranderen we namelijk de waarde van de variabele van ongedefinieerd naar 1.

Goed, we hebben nu een startpunt voor de variabele die we in de conditie van de lus gebruiken. Dus kunnen we een route plannen. Waarom hebben we als startpunt het getal 1 gekozen? Precies, omdat we eerst het getal 1 willen laten afdrukken door het programma. In onze conditie staat het getal 500. Dat is dus ons doel. We moeten onze variabele `getal` daar naartoe brengen door deze te veranderen. Aangezien we na het getal 1 het getal 2 en daarna het getal 3 enzovoorts willen laten afdrukken is het zinvol om telkens 1 bij de waarde van de variabele `getal` op te tellen. Dat doen we in ons hoofd feitelijk ook als we aan het tellen zijn. Dit zien we ook terug in het PSD in het statement `getal := getal + 1`.

Het statement `getal := getal + 1` kan heel verwarrend overkomen. Wat als we het verwoorden? Dan staat er "De variabele `getal` wordt de variabele `getal` plus 1." Hoe kan iets zichzelf worden? Precies, dat kan niet en dat gebeurt ook niet. Als we een variabele weer als een doos zien gebeurt er het volgende.

> Het programma pakt uit de doos de waarde van de variabele, telt er 1 bij op en stopt vervolgens het resultaat van de optelling terug in de doos.

Dit kunnen we als volgt verwoorden:

> De nieuwe waarde van de variabele `getal` wordt de oude waarde van de variabele `getal` plus 1.

Onze initiële verwoording van het statement `getal := 1` kan dus ook beter, namelijk: "De nieuwe waarde van de variabele `getal` wordt 1."

Programmeurs zeggen bij het statement `getal := getal + 1` vaak: "We verhogen de waarde van de variabele `getal` met 1."

Waarom hebben we als conditie `getal > 500` gekozen? Dit moet omdat 500 het laatste getal is dat we willen laten afdrukken en we onze variabele `getal` steeds met 1 verhogen. Omdat we een totdat-lus gebruiken zal de lus eindigen bij het getal 501 want dat is het eerste getal dat we bereiken na 500 door telkens 1 op te tellen bij de variabele `getal`. 500 wordt dus nog wel afgedrukt maar 501 niet.

Misschien is dit allemaal nogal overweldigend. Geen paniek. Zoals gezegd: "Oefening baart kunst!" Als je er niet uitkomt stel dan vooral vragen. Dat mag natuurlijk aan je leraar maar ook aan klasgenoten die misschien al goed zijn in programmeren.

opgave 4.4

Waar of niet waar?

a) Een lus is een handige manier om je veel saai programmeerwerk uit handen te nemen.
b) Een oneindige lus in een PSD is een lus die de computer uit blijft voeren totdat een mens de computer stopt.
c) Code na een oneindige lus zal een programma nooit uitvoeren.
d) Als je wilt dat een totdat-lus eindigt moet de conditie van de lus ooit waar worden.
e) Variabelen kun je van waarde veranderen.
f) Als je een variabele een nieuwe waarde geeft is de oude waarde verdwenen.
g) De beste manier om statement `a := 1` te verwoorden is: "De variabele `a` wordt 1."

opgave 4.5

Maak een PSD dat de getallen 2, 4, 6, 8 en 10 afdrukt in een pop-up. Je moet een totdat-lus gebruiken.

opgave 4.6

Maak een PSD dat de getallen 10 tot en met 1 afdrukt (dus in aflopende volgorde) in een pop-up. Je moet een totdat-lus gebruiken.

opgave 4.7

Maak een PSD dat de getallen 1, 2, 4, 8, 16, 32, 64, 128 en 256 afdrukt in een pop-up. Je moet een totdat-lus gebruiken.

opgave 4.8

Maak een PSD dat de getallen uit opgave 4.7 in aflopende volgorde afdrukt in een pop-up. Je moet een totdat-lus gebruiken.

opgave 4.9

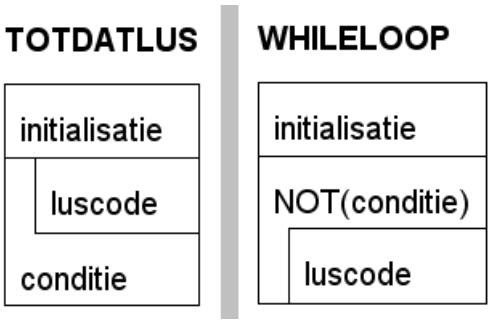
Maak een PSD dat de resultaten van de tafel van 7 afdrukt in een pop-up. Je moet een totdat-lus gebruiken.

5. andere soorten lussen

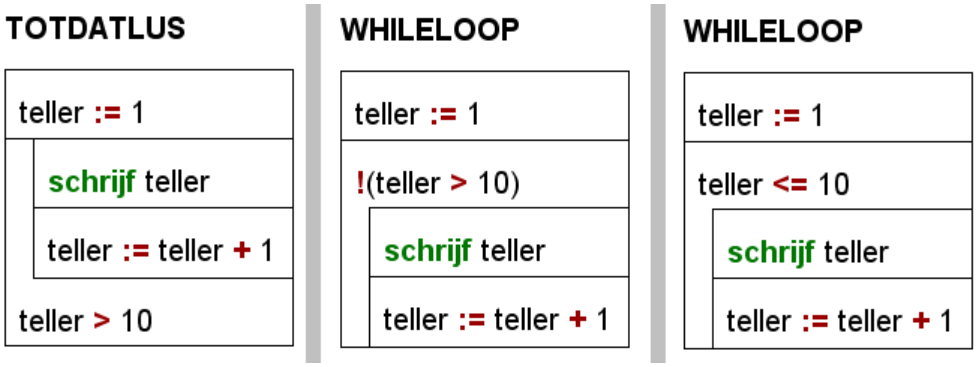
Zoals we al eerder hebben gezegd, bestaan er ook nog andere lussen dan de totdat-lus. Deze zijn voor beginnende programmeurs minder intuïtief, maar voor ervaren programmeurs juist intuïtiever. Sterker nog, er zijn programmeertalen die de totdat-lus **niet** ondersteunen. Het is daarom belangrijk dat je met name leert werken met de zolang-lus (while-loop).

Waar een programma bij een totdat-lus de lus-code blijft uitvoeren totdat de conditie waar is, zal een programma bij een while-loop de lus-code blijven uitvoeren zolang de conditie waar is. De while-loop doet met betrekking tot de conditie dus feitelijk het omgekeerde van de totdat-lus.

Gelukkig kun je een totdat-lus heel makkelijk omschrijven naar een while-loop. Als je while-loops dus moeilijk vindt, dan kun je altijd een totdat-lus maken en die vervolgens omschrijven. Het omschrijven is namelijk niet moeilijk. In Figuur 20 zie je hoe je een totdat-lus verandert in een while-loop. In Figuur 21 zie je een concreet voorbeeld van het omschrijven. De meeste programmeertalen snappen het woord NOT niet, in plaats daarvan moeten we het uitroepteken gebruiken. Meestal schrijven we de conditie om naar iets zonder het uitroepteken, omdat dat makkelijker leest. Dat is de laatste stap in Figuur 21.



Figuur 20: omschrijven totdat-lus naar while-loop

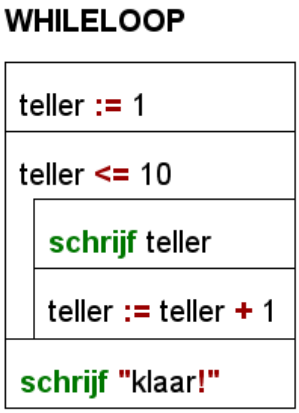


Figuur 21: praktijk voorbeeld omschrijven

Herinner je je nog dat de lus-code van een totdat-lus even vaak uitgevoerd wordt als dat de conditie ervan gecontroleerd wordt? Bij een while-loop is dit **niet** het geval. Hier wordt de conditie één keer meer gecontroleerd dan dat de lus-code uitgevoerd wordt. Dit kun je zien als we een mogelijke executie van de while-loop uitschrijven:

1. kijk of de conditie waar is
2. de conditie is waar, dus voer de lus code uit
3. kijk of de conditie waar is
4. de conditie is waar, dus voer de lus code uit
5. kijk of de conditie waar is
6. de conditie is waar, dus voer de lus code uit
7. kijk of de conditie waar is
8. de conditie is niet waar, dus voer het eerste stuk code na de lus-code uit

Je kunt zien dat, wanneer een while-loop eindigt, de executie van het PSD dus niet verder gaat na de conditie maar na de lus-code. Wanneer in het programma van Figuur 22 de conditie dus onwaar wordt (het is een while-loop, dus de lus eindigt als de conditie onwaar wordt, precies omgekeerd ten opzichte van de totdat-lus!) gaat het programma niet verder bij het statement `schrijf teller` maar bij het statement `schrijf klaar`. Het programma slaat de lus-code over als de lus eindigt. Dit maakt dat voor veel mensen de while-loop moeilijker te begrijpen is dan de totdat-lus.



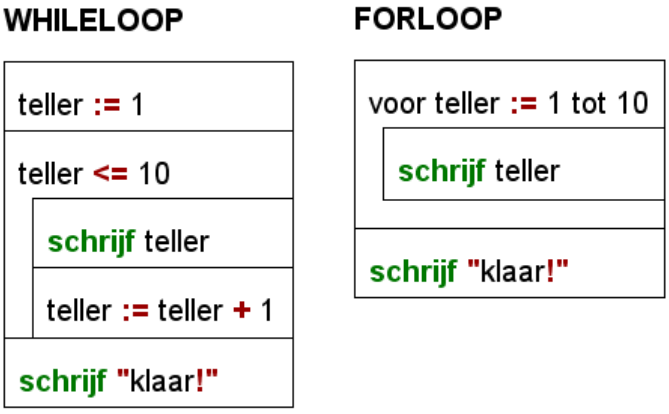
Figuur 22: while-loop met code na de lus-code

Ook hier geldt weer: "Oefening baart kunst!"

opgave 5.1

Maak een PSD voor de opdrachten in opgaves 4.5, 4.6 en 4.7. Gebruik nu echter een while-loop en zorg dat het programma als laatste een pop-up geeft met daarin de tekst "klaar!".

De while-loop heeft ook nog een verkorte variant die de for-loop heet. Deze twee lussen werken op precies dezelfde manier alleen is de schrijfwijze van de for-loop korter. In Figuur 23 zie je ze naast elkaar.



Figuur 23: while- en for-loop naast elkaar

Je ziet dat er bij de for-loop `tot 10` staat. Dit is niet helemaal correct want het zou eigenlijk `tot en met 10` moeten zijn. Helaas vinden computers dit niet fijn om te lezen omdat er spaties in staan. Ook `t/m 10` vinden ze niet fijn want ze zien / als een deelstreep. Vandaar dat we in een PSD `tot` gebruiken al is dit feitelijk niet correct. Bij de meeste programmeertalen speelt dit probleem niet.

Ook met de for-loop oefenen we nog een beetje. Je hoeft feitelijk nooit een for-loop te gebruiken als je dat niet wilt. Je kunt het altijd met een while-loop oplossen. Er zijn geen populaire programmeertalen die de while- en for-loop niet ondersteunen.

De for-loop in Structorizer is verre van ideaal. Je kunt er bijvoorbeeld niet mee aftellen of de teller steeds met 2 verhogen. Daarom ga je nu niet oefenen met de for-loop. Echte programmeertalen kennen deze tekortkomingen met betrekking tot de for-loop niet en programmeurs gebruiken deze loop dan ook vaak vanwege de compactere schrijfwijze.

6. wat heb je geleerd

Je hebt zojuist een flinke stap voorwaarts gezet in het leren programmeren als je dat nog niet kon. Je hebt de volgende programmeerconcepten leren kennen en ermee gewerkt:

- sequentie
- statement (instructie)
- selectie
- conditie
- geneste selectie
- herhaling (lussen)
- lus-code
- totdat-lus (repeat-until-loop)
- eindigen van een lus
- geneste lus
- variabele
- oneindige lus
- initialisatie
- toekenning
- while-loop (zolang-lus)
- for-loop

En natuurlijk heb je met Structorizer leren werken en je weet nu hoe je een PSD moet lezen en maken.