BASIS PROGRAMMEREN

De Nittis Massimo i.o SyntraPXL 2023

1. Inleiding

Als het programmeren je wat zegt dan heb je waarschijnlijk nú al wilde plannen over een programma dat je zou willen schrijven. Later zal het dan zo zijn dat jouw klant een wild idee heeft en vraagt of jij zijn idee kan omzetten in een programma. Vaak is het echter zo dat het niet onmiddellijk werkt zoals jij of jouw klant in gedachten had. Het werk van een programmeur komt heel goed overeen met dat van een architect:

Het eerste wat de architect zal moeten doen, is luisteren naar de bouwheer: wil hij één of twee garages, zijn het liefhebbers van een tuin, welke stijl wensen ze,...? Je begrijpt best dat dit een heel belangrijke eerste stap is, want als architect en bouwheer elkaar verkeerd begrijpen, kan dit ernstige gevolgen hebben voor het gebouw.

Vervolgens zal de architect zich terugtrekken en nadenken over hoe hij de wensen van de klant kan realiseren. Hij tekent een plan en maakt eventueel een schaalmodel van het huis.

De architect trekt met zijn plan en/of schaalmodel van het huis naar de bouwheer. Ze overlopen alles nog eens en als blijkt dat de klant dit of dat anders wil, gaat hij terug aan de tekentafel zitten (dus terug naar de vorige stap). Komen ze echter tot een akkoord, dan kan de bouwfase starten (de volgende stap). Merk dus op dat er soms - ook bij programmeren - moet worden teruggekeerd naar een vorige stap.

De bouwwerken worden niet uitgevoerd door de architect zelf, maar door een aannemer. Het is echter wel de taak van de architect om te controleren of alles wel gebouwd wordt zoals op het plan wordt aangegeven. Als de metselaar de ruimte voor het raam te klein maakt, dan zal het raam er gewoon niet in passen.

Soms gebeurt het dat de bouwheer toch niet helemaal tevreden is en wil bijsturen (bv. een extra veranda of een bijkomende badkamer) en dan begint het proces helemaal opnieuw.

Het grote verschil tussen programmeren en bouwen is dat je programmacode makkelijker kan uitbreiden en veranderen dan bij een bouwwerk. Stel je voor dat als het huis helemaal klaar is, blijkt dat ze de fundamenten vergeten zijn... Makkelijker betekent echter niet 'gemakkelijk', dus ook bij programmeren is die eerste stap van lezen en begrijpen héél belangrijk! Hoe beter uitgedacht, des te makkelijker je later je programma kan uitbreiden en veranderen.

2. Deelproblemen

Bij eenvoudige problemen lijkt de oplossing soms heel makkelijk. Toch is het belangrijk te leren één probleem op te splitsen in deelproblemen. Dat maakt het veel makkelijker om een complex probleem te kunnen oplossen. Bovendien is het zo dat als je dit goed kan, je de deelproblemen kan uitbesteden aan meerdere personen.

Heel wat zaken hebben niet rechtstreeks iets te maken met programmeren, maar wel met het opsplitsen van deelproblemen:

1. Het bouwen van een huis is een mooi voorbeeld van het opsplitsen van deelproblemen, daar veel deelproblemen worden uitgevoerd door aparte personen. Zo heb je de metselaar, de vloerder, de loodgieter, de elektricien,... Uiteindelijk bekom je een volledig afgewerkt huis.
2. Het maken van spaghetti.
3. Lekke band op de weg? Dan moet deze vervangen worden.
4. Een groepswerk van geschiedenis.

Iemand die heel goed deelproblemen kan onderscheiden, heeft zeker een groot voordeel bij het aanpakken van grotere programmeerprojecten.

3. Klassen en objecten

Bij een objectgeoriënteerde programmeertaal leunt men sterk aan bij de wereld zoals wij die ook ervaren: namelijk een wereld met een verzameling van objecten. **Een object is echter iets concreet: de vierkante witte tafel is een ander object dan die ronde blauwe tafel.**

In de programmeertaal wil men echter ook de blauwdruk van een object kunnen vastleggen. Als men het heeft over de '**blauwdruk**', dan spreekt men over de klasse van een object. In het voorgaande geval is de klasse 'Tafel', terwijl de concrete tafels objecten zijn.

Zo ook zijn Bill Gates van Microsoft en Steve Jobs van Apple allebei objecten, maar het zijn objecten van dezelfde klasse, namelijk 'Persoon'. De blauwdruk van die klasse is dan het feit dat een Persoon een voornaam, een familienaam en een bedrijf heeft

3. Variabelen & Constanten

3.1 Variabelen:
Soms heb je een bepaalde (berekende) waarde (value) doorheen de programmacode meer dan één keer nodig. In dat geval kun je – i.p.v. die waarde steeds opnieuw te berekenen – de waarde opslaan in een variabele en dan gewoon de naam van de variabele gebruiken. Bij een complexe berekening kan je d.m.v. variabelen deze berekening opsplitsen in tussenstappen, waarvan je elk tussenresultaat kan opslaan in een aparte variabele. Een variabele is dus een benoemde geheugenlocatie waar gegevens opgeslagen kunnen worden.

We noemen 'som' een variabele. Deze variabele is nodig omdat we ergens een plaats willen in het geheugen om het resultaat van 'getal1 + getal2' in op te slaan. Wij hoeven als programmeur niet te weten waar dat dan precies is, daar wij enkel de naam moeten onthouden.

3.2 Constanten
Bepaalde waarden blijven gedurende gans het programma – onafhankelijk van wanneer of hoe het uitgevoerd wordt – dezelfde. Zo is het getal π een constante, alsook de datum waarop Kerstmis valt. Als je in programmeren gebruikt maakt van een constante, dan kan je de waarde niet meer veranderen.

Op die manier worden fouten vermeden en wordt het onderhoud van de programmacode eenvoudiger. Ook hier is vaak een gegevenstype van toepassing, dezelfde als bij variabelen.

4. Controlestructuren & loops

Via controlestructuren kan je de manier waarop je programma wordt uitgevoerd beïnvloeden. Waarbij we gebruikmaken van zaken die al bepaald zijn, zoals de constanten en variabelen.

Nassi-Schneidermann diagrammen, kortweg NS-diagrammen of ook wel structogrammen genoemd, zijn een goed middel om algoritmen voor te stellen, doordat het automatisch leidt tot gestructureerd werken. Loops zorgen voor de iteratie, oftewel de herhaling tot aan een bepaalde voorwaarde wordt voldaan. Er bestaan verschillende soorten loops, met elk hun eigen doel;

5. Operatoren

5.1 Toekenningsoperator

Met de toepassings-of toekennings operator (vaak voorgesteld door het gelijkheidsteken =) kan je

1. (nieuwe) waarden toekennen aan variabelen;
2. (nieuwe) waarden toekennen aan objecteigenschappen;

5.2 Rekenkundige operatoren

De eerste belangrijke uitdrukkingen zijn de rekenkundige operatoren: hoe kan je getallen (of de variabelen waar ze in zitten opgeslagen) optellen, aftrekken, delen of vermenigvuldigen? In heel wat programmeertalen wordt dit gedaan met respectievelijk +, -, / en *.

5.3 Relationele operatoren

Met de relationele operatoren kan je relaties tussen waarden of variabelen testen en als uitkomst geven ze steeds 'waar' of 'onwaar'. Zo heb je groter dan (>), groter dan of gelijk aan (>=), kleiner dan (<) en kleiner dan of gelijk aan (<=).

Andere belangrijke relationele operators zijn gelijk aan en niet gelijk aan, maar de syntax kan per taal anders zijn. Zo wordt in VBA gebruik gemaakt van respectievelijk = en <> en in PHP en Java van == en !=.

5.4 Logische operatoren

Soms moeten meerdere voorwaarden voldaan zijn: dat kan door het werken met 'en' of met 'of': in heel wat talen wordt dat voorgesteld door 'and' en 'or'. Ook belangrijk is het 'tegenovergestelde' van een voorwaarde verkrijgen, wat je PHP met ! doet en VBA met Not.

6. Commentaar

Een van de belangrijkste dingen bij het programmeren, is commentaar. Commentaar wordt gebruikt om aan te geven wat een stuk code doet en maakt je code stukken duidelijk. Het lijkt alsof dat alleen maar tijd en energie kost. Als je echter na een tijdje jouw code opnieuw bekijkt of als iemand anders jouw code moet bekijken, dan kan die commentaar een heel welgekomen hulp zijn (zeker bij duizenden lijnen code).