Pascal
en microcomputers

H. J. C. Otten

Pascal is een belangrijke programmeertaal. Niet alleen is Pascal belangrijk omdat het een veel gebruikte computertaal is, maar omdat hierin voor het eerst de denkbeelden over gestructureerd programmeren zijn geïmplementeerd.
Pascal is ontworpen in 1970 als hulpmiddel bij het onderwijs in de informatica door Niklaus Wirth en is vervolgens zeer populair geworden op universiteiten en scholen. Onvermijdelijk gevolg hiervan is dat Pascal ook als programmeertaal bij professionele automatiseringsprojekten meer en meer wordt gebruikt. Vooral bij microcomputergebruikers is de belangstelling voor Pascal groot. In dit artikel wil ik mijn eigen ervaringen met Pascal bespreken met een aantal implementaties en de taal zelf.

Waarom Pascal?

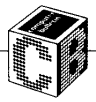
Een programmeertaal is een hulpmiddel voor mensen om computers te laten doen wat deze mensen willen. Programmeertalen zijn kunstmatige talen met een bekrompen opvatting over wat wel en niet is toegestaan. Vooral over kleinigheden kunnen meningsverschillen ontstaan tussen mens en computer. De mens bedoelt het meestal wel goed, maar vertelt het de computer niet ondubbelzinnig, met als gevolg dat de computer niet doet wat de mens wil. Ook is het voor een mens niet gemakkelijk om een sluitende oplossing voor een probleem te geven.
Na een aantal jaren met computers te hebben gewerkt is het voor mij, en iedere gewetensvolle programmeur duidelijk, dat programmeren niet alleen voor beginnelingen moeilijk is. We kunnen bij het programmeren daarom alle hulp gebruiken. Gestructureerde werkmethode en een programmeertaal waarin die methoden op een natuurlijke manier tot uiting komen, zijn goede hulpmiddelen.

Structuur

Omdat programmeren moeilijk is, geldt hiervoor wat voor alle gecompliceerde problemen geldt: gebruik heldere, kleine en simpele stappen om het probleem op te lossen. Als een oplossing van een probleem er ingewikkeld uitziet is die oplossing zeer waarschijnlijk niet goed en beperkt toepasbaar. In ieder geval is de oplossing niet meer bruikbaar als het probleem zelf verandert. Juist dat laatste is een veel voorkomende zaak en leidt tot enorm kostbaar onderhoud aan programmatuur.
We komen al een aardig eind op de goede weg als we onze werkwijze structureren. De informatica is nog een jonge wetenschap en de eerste bruikbare bijdragen daarvan waren de ideeën over gestructureerd programmeren. Vooral een Nederlander, professor Edsger Dijkstra uit Eindhoven, heeft hier fundamentele bijdragen geleverd. Alleen en samen met andere informatici heeft hij zijn ideeën onder andere vastgelegd in twee boeken: „Structured programming" en „Systematic programming: an introduction". Dit zijn zeer interessante boeken waarin de ideeën over gestructureerd programmeren, correctheidsbewijzen, top-down-ontwikkelingen enz. aan bod komen. Maar de nieuwsgierige lezer wil ik wel waarschuwen dat het boeken zijn, geschreven door en voor informatici, met een sterk wiskundig karakter.

Ontstaan van Pascal

Een van de informatici die nauw betrokken was bij het ontwikkelen van de ideeën over gestructureerd programmeren was Niklaus Wirth. Omdat de oudere programmeertalen niet erg geschikt waren om de gestructureerde technieken helder aan studenten te tonen, heeft Wirth een nieuwe programmeertaal voor onderwijsdoeleinden ontworpen en de taal de naam gegeven van een bekend wiskundige: Blaise Pascal. Wirth had bij het ontwerpen van Pascal het doel een taal te definiëren om gestructureerde programmeertechnieken te onderwijzen. Het was niet het hoofddoel een efficiënte en complete productietaal, zoals Fortran en Cobol wel zijn, te ontwerpen, maar een helder en kleine kunstmatige taal voor het ontwikkelen van gestructureerde algorithmen en duidelijke datastructuren. Pas jaren later heeft Wirth een taal ontworpen gebaseerd op dezelfde ideeën, maar met een sterk productiekarakter: Modula-2 (zie het artikel over Lilith in RB van april 1982). Pascal



begon dus met een handicap als productietaal.

Pascal nu

Het is onvermijdelijk dat een programmeertaal, die studenten leren op school en universiteit, doordringt in een productie-omgeving. Mij is het ook zo vergaan. Op de Vrije Universiteit heb ik leren programmeren met Pascal als hulpmiddel. De Vrije Universiteit heeft, zoals vele universiteiten, het Unix-operating systeem op een PDP-11-minicomputer toegepast en door Andrew Tanenbaum en Johan Stevenson is een uitstekende Pascal-implementatie gemaakt: de VU-Pascal-compiler.
Als ik nu de keuze heb om een programma te schrijven in de taal die mij het beste bevalt, dan kies ik voor Pascal. Zo heb ik al kennis gemaakt met vele implementaties van Pascal: op mainframes (VS-Pascal onder VM-CMS op IBM-computers), op minicomputers (Pascal-1000 onder RTE op HP1000-computers) en op microcomputers. De implementaties op microcomputers zal ik later nader bespreken. Als ik niet de vrije keuze in programmeertaal heb, dan speelt Pascal nog steeds een rol. Voordat het programma in de productietaal wordt gecodeerd, worden de algorithmen van het programma door mij eerst op papier ontwikkeld in een Pascal-achtige taal: pseudo-code.
Vooral de stormachtige ontwikkelingen rondom de microcomputer hebben de verspreiding van Pascal bevorderd. De chaos in de implementaties van Basic maken Basic totaal ongeschikt als een taal om programma's in te schrijven, die op andere computers zonder veel problemen zijn over te zetten. De taal zelf is overigens niet erg geschikt om grote en complexe programma's in te schrijven. Alhoewel gestructureerd programmeren met elke programmeertaal mogelijk is, is de hulp van Basic daarbij minimaal. Dit betekent, dat veel te veel aan de programmeur wordt overgelaten en die heeft het met het programmeren al moeilijk genoeg.

Problemen met Pascal

De al genoemde chaos in de implementaties van Basic zijn een ernstige handicap voor het gebruik ervan. Helaas is Pascal niet vrij van een dergelijke chaos, gelukkig wel in een veel geringere mate. Daar zijn twee hoofdoorzaken voor aan te wijzen. Ten eerste is Pascal niet zonder meer gestandaardiseerd. In het boek „Pascal manual and Report" van Jensen en Wirth wordt Pascal wel gedefinieerd, maar er zijn nog vele zaken opengelaten. Later is er een zogenoemde ISO-standaard voor Pascal verschenen, die vele onduidelijkheden oplost. Een Pascal-implementatie, die zich aan de ISO-standaard conformeert, wordt meestal standaard-Pascal genoemd. Op het moment wordt er gewerkt aan een ANSI-standaard voor Pascal, zoals die ook bestaat voor Fortran en Cobol. Ten tweede is standaard-Pascal een erg kleine taal, waaraan een groot aantal belangrijke mogelijkheden ontbreken en dat heeft ertoe geleid dat vrijwel elke implementatie in ieder geval iets toevoegt aan standaard-Pascal of er van afwijkt. Nu zou dat geen probleem zijn als iedereen dezelfde toevoegingen zou hanteren, maar iedereen vindt helaas weer opnieuw het wiel uit voor de uitbreidingen en dàt wiel is niet altijd rond...

Problemen in standaard-Pascal

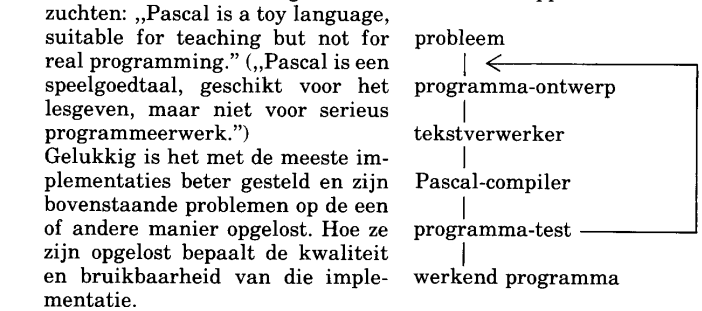
In een aantal artikelen, aan het einde van dit artikel genoemd, is door vooraanstaande informatici gewezen op problemen in standaard-Pascal. Om de hieronder genoemde problemen te begrijpen is wel een goede kennis van Pascal nodig:

1. Arrays hebben in Pascal een vaste grootte, vastgelegd voor het compileren. Dit maakt bijvoorbeeld het gebruik van een bibliotheek van wiskundige standaardroutines onmogelijk en bemoeilijkt het werken met strings. Als ik bijvoorbeeld een procedure voor een complexe matrix-operatie nodig heb en er zijn twee matrices in het programma met verschillende dimensies, dan ben ik gedwongen tweemaal dezelfde procedure voor verschillende dimensies in het programma op te nemen.
2. Variabelen die lokaal in een procedure zijn gedeclareerd, bestaan alleen tijdens de activatie van die procedure en het lokaal zijn beschermt de variabele voor de rest van het programma. Een variabele die eigenlijk lokaal voor een procedure is, maar waarvan de waarde behouden moet blijven tussen verschillende activaties van de procedure, moet globaal worden gedeclareerd en is niet beschermd.
   Een voorbeeld is een procedure die een regel naar een regelprinter stuurt en daarbij een regelteller bijhoudt om de papierindeling te regelen. De regelteller is logisch gezien een lokale variabele, maar kan niet lokaal worden gedefinieerd.
3. Pascal programma's moeten als geheel worden gecompileerd. Het ontwikkelen van grote programma's wordt daardoor bemoeilijkt en in ieder geval tijdrovend en het opbouwen van bibliotheken is onmogelijk.
4. Het is niet mogelijk declaraties in een willekeurige logische volgorde te te doen en de declaratie van constanten is primitief. Het initialiseren van constanten in arrays of records is niet mogelijk.
5. Als een variabele in een Case-statement niet voorkomt in de opgenoemde lijst, dan is het resultaat onvoorspelbaar, met andere woorden er ontbreekt een „otherwise"-tak in een Case-vertakking.
6. De in/uit-mogelijkheden van Pascal zijn uiterst beperkt. Alleen sequentiële files zijn te gebruiken, random access en indexed files bestaan niet in Pascal.
7. Interactief met een terminal werken op karakterbasis is niet mogelijk.
8. Gewoonlijk is de sterke typechecking van Pascal een uitstekende bescherming. Bij het implementeren van primitieve in/uit-operaties is het echter een handicap, die alleen met implementatie-afhankelijke trucs zijn op te lossen.
9. Het is in de praktijk gebleken dat strings zo vaak voorkomen, dat het type string en een aantal goede stringfuncties eigenlijk tot het standaard-repertoire van Pascal zouden moeten horen.

► De bovenstaande lijst met problemen deed Brian Kernighan verzuchten: „Pascal is a toy language, suitable for teaching but not for real programming." („Pascal is een speelgoedtaal, geschikt voor het lesgeven, maar niet voor serieus programmeerwerk.")
Gelukkig is het met de meeste implementaties beter gesteld en zijn bovenstaande problemen op de een of andere manier opgelost. Hoe ze zijn opgelost bepaalt de kwaliteit en bruikbaarheid van die implementatie.

Werken met Pascal

Pascal is een hogere programmeertaal en voordat een programma, dat in Pascal is geschreven, kan worden gestart moet het worden vertaald naar machine-code.
Dit betekent dat het draaien van een programma een aantal voorbereidende stappen vereist:



Het programma begint als een met de computer op te lossen probleem. Eerst wordt het programma ontworpen door de programmeur die de programma-tekst met behulp van een tekstverwerker „editor" de computer invoert.
De Pascal-compiler vertaalt de tekst, „source", naar machine-code en vervolgens kan het programma, „code", worden getest. Als er een fout optreedt moet het proces worden herhaald, anders hebben we een werkend programma. In deze ontwikkelcyclus neemt de compiler alleen deel als vertaler. De compiler voert in tegenstelling tot bijvoorbeeld een Basic-interpreter het programma niet uit! Al deze stappen samen kosten aardig wat tijd en machine-capaciteit. Het resultaat is echter meestal beter dan met Basic-interpreters, want als het programma eenmaal correct is vertaald, loopt het programma een stuk sneller. Dat het programma beter leesbaar is en daardoor beter is te onderhouden en uit te breiden mag vanzelf spreken.