Pascal uitgediept

Herman Post
MSX Computer & Club Magazine nummer 66 - maart / april 1994

Scanned, ocr'ed and converted to PDF by HansO, 2001

Deze keer wordt de duistere kant van Pascal wat in het zonnetje gezet. Een aantal variabelen, dat in Pascal is ingebouwd, maar waar de handleiding bijna geen informatie over verschaft, wordt hier genoemd en wat toegelicht. Veel experimenteren is de boodschap.
Iedereen die in Pascal programmeert weet dat variabelen een onmisbaar deel van een programma zijn. Ze zijn meestal dan ook erg eenvoudig te declareren. Het is daarom wat vreemd dat veel programmeurs niet op de hoogte zijn van een aantal voorgedefinieerde variabelen. Hiervan zijn de variabelen pi en maxint goede voorbeelden. Deze kunnen worden gebruikt zonder ze te initi-aliseren. Wijzigen van die variabelen kan, en mag ook, maar het handboek raadt dit ten sterkste af. Het brengt een hoop verwarring mee, en uw programma's worden er niet duidelijker door. Ik ben het er voor de hierboven genoemde variabelen helemaal mee eens. Er zijn echter ook variabelen die in het handboek vermeld staan, waarbij zo weinig informatie wordt gegeven, dat je, als goedwillende hobbyist, toch wel wat nieuwsgierig •wordt, en er wat verder naar gaat kijken. Ik bedoel hier de verschillende pointers zoals er zijn:
AuxInPtr, AuxOutPtr, ConStPtr, ConInPtr, ConOutPtr, ErrorPtr, HeapPtr, LstOutPtr, RecurPtr, UsrInPtr, UsrOutPtr en de variabele BufLen. Het wordt nog vreemder als je tegen een variabele aanloopt die in het geheel niet in het handboek voorkomt zoals CBreak.

Pointers

Het gebruik van de verschillende pointers is nog wel duidelijk. Vooral omdat de HeapPtr en de ErrorPtr redelijk van informatie zijn voorzien en het gebruik van de overige pointers is hieruit ook wel af te leiden. In het kort komt het er op neer dat iedere pointer verwijst naar een routine. Zo wijst de ErrorPtr naar een routine die gebruikt wordt als er een fout optreedt. Het gaat hierbij om de run-time- en I/0-fouten. Op dat moment wordt naar het adres gesprongen dat in ErrorPtr staat. Het idee daar achter is dat u een eigen foutafhandelingsroutine schrijft, en de ErrorPtr daar naartoe laat wijzen. Veel schiet u er echter niet mee op, want nadat deze zelfge-schreven routine is afgewerkt, verlaat Pascal alsnog zijn eigen programma omdat er een fout was geconstateerd. Het voordeel is wel dat u bij grafische programma's eerst kunt terugspringen naar een tekstscherm en dan het programma verlaten, (vergelijk dit eens met de jANSI tip uit nr 63 biz 29) Wat betreft de Aux, Con, Lst en Usr pointers is dit op eenzelfde manier gedaan. Allemaal bevatten ze een adres van een routine die deze verwerkt. In het kader onderaan kunt u zien wat de routines, die aangeroepen worden, moeten doen en hoe u deze zelf opnieuw kunt programmeren. Met dank aan Albert Meek, die deze informatie netjes bij elkaar had gezet in het clubboekje van MSX computer Club Enschede. De HeapPtr verwijst niet

naar een routine, maar naar de eerste geheugenbyte van de heap. U kunt deze dus op een andere plaats zetten en zo geheugen vrijmaken tussen het data-segment en de heap. Omdat dit erg complexe materie is, wordt dit in alle boeken afgeraden. Wilt u er toch iets meer over weten, dan vindt u in het boek 'turbo pascal voor gevorderden' nog de meeste informatie.

Pointer	Toepassing	**Statement omschrijving**
AuxinPtr	Leest een karakter van het aux: device	FUNCTION LeesAux: CHAR;
AuxOutPtr	Schrijft een karakter naar het aux: device	PROCEDURE SchrijfAux(ch : CHAR);
ConinPtr 1)	Leest een karakter van het toetsenbord (con:)	FUNCTION LeesCon : CHAR;
ConOutPtr 2)	Schrijft een karakter op het scherm (con:)	PROCEDURE_SchrijfCon(ch CHAR);
ConStPtr	Kijkt of er een karakter in de keyboardbuffer staat	FUNCTION KarakterGereed: BOOLEAN;
ErrorPtr	Geeft foutnummer plus buHibehorend adres -	PROCEDURE Error(foutnr BYTE; adres : INTEGER);
LstOutPtr	Zendt een karakter naar de printer (lst:)	PROCEDURE Schrijflist(ch CHAR);
UsrInPtr)	Leest een karakter van het usr: device	FUNCTION LeesUsr : CHAR;
UsrOutPtr	Schrijft een karakter naar het usr: device	PROCEDURE SchrijfUsr(ch : CHAR);

Opmerkingen
1) UsrInPtr en ConInPtr wijzen naar dezelfde routine
2) UsrOutPtr en ConOutPtr wijzen naar dezelfde routine

Gebruik

Nu even over het gebruik van deze routines, want dat maakt het pas leuk. Het is de gewoonte van Pascalgebruikers om write en writein te gebruiken als WRITE('hallo ') ; Maar Pascal leest dit als WRITE (Con, 'hallo') ;. Hierbij zien we direct dat de data naar het Con device wordt gestuurd. Op dezelfde manier wordt er iets naar de printer gestuurd.

```
WRITE (Lst, 'hallo') ;
```

zet de tekst op de printer. tenminste, volgens de handleiding. Door een bug in Pascal werkt dit niet. De meest gebruikte truc is om LST te declareren als text, en boven in het programma de regels
:

```
ASSIGN(LST,'Lst:');
REWRITE(LST);
```

op te nemen en alles werkt naar behoren. Hier wordt simpelweg DOS ingeschakeld om het daadwerkelijke printen voor ons uit te voeren. De andere mogelijkheid is om zelf een routine te schrijven die een karakter naar de printer stuurt, en de LstOutPtr naar deze routine te laten wijzen. Als we dan schrijven:

```
WRITE(LST,'op de printer');
```

wordt dit keurig naar de printer gezonden. Op dezelfde manier kunnen we zaken naar de aux en usr devices sturen. Zoals in het kader staat opgemerkt zijn de con en de usr devices op hetzelfde adres ingesteld. Met andere woorden:alles wat via de usr wordt afgedrukt, komt op dezelfde manier op het scherm te staan als dat we dit normaal zouden zien met WRITE. Nu gaan we het usr device afbuigen naar een eigen routine, die tekst op het scherm zet. Deze maakt geen gebruik van de BDOS en is bijna twee keer zo snel als standaard Pascal. Er wordt nog wel gebruik gemaakt van de BIOS, dus het kan nog sneller, maar dat laat ik graag aan machmetaalmensen over.

```
PROCEDURE Char0ut(ch : CHAR);

  INLINE ($3A/ch/            {LDA,(ch)}
          $FD/$2A/$CO/$FC/   {LD IY,(FCCO) }
          $DD/$21/$A2/$00/   {LD IX,OOA2 adres chput }
          $CD/$1C/$00)       {CALL 001C calslt }
END;
```

In ons programma nemen we nu op:

```
UsrOutPtr:=ADDR(CharOut);
```

Om nu tekst op het scherm te gaan weergeven schrijven we

```
WRITE(USR,tekst);
```

De tekst wordt nu via de BIOS afgedrukt en wordt niet eerst door de BDOS gehaald. Het gevolg is een veel snellere schermopbouw.

Redirection en pipelining

De begrippen hierboven zijn afkomstig van DOS 2.xx. In de programmeertaal C wordt dit ondersteund, in Pascal niet. Door echter de pointers om te gooien kan het wel. Een goede handleiding, en wat ervaring met machinetaal, is hier echter wel op zijn plaats. Omdat het mij aan die handleidingen niet ontbreekt, maar wel aan de nodige machinetaal-kennis, heb ik dit niet volledig goed werkend gekregen. Dit wil zeggen dat ik wel een programma heb gemaakt dat netjes een file op de juiste manier verwerkt, maar nu wordt het toetsenbord niet meer goed uitgelezen. U kunt dit programma op de disk bij dit magazine terugvinden onder de naam RIGHT.COM en R1GHT.PAS. Het gebruik is eenvoudig (alleen DOS2!):

```
DIR | RIGHT of
```

```
RIGHT < FNAAM.EXT of
RIGHT < FNAAM.EXT > NEWF.EXT
```

Het gevolg is dat de directory of een tekstfile rechts uitgelijnd op uw scherm verschijnt. Het is niet bedoeld als een programma dat erg nuttig is, maar enkel om de mogelijkheid te demonstreren. Dit is ook de reden dat ik het toetsenbord niet netjes aan het werk kreeg. Dit zou ik simpel kunnen doen door geen omleiding te gebruiken, maar in normaal Pascal alles in te lezen en dit naar het scherm, of naar de juiste file weer uit te voeren. Ter verduidelijking plaats ik op het scherm wat als invoer en wat als uitvoer wordt gebruikt zodat u zelf allerlei toepassingen kunt verzinnen. De invoer wordt in dit programma karakter voor karakter gedaan. Dan wordt er wat gestoeid met de CR/LF-code zodat deze eruit wordt gefilterd en de letters worden in een string geplaatst. Het voortdurend op CR/LF controleren is gedaan omdat de nieuwe string weer met een WRITELN wordt weggeschreven. Het WRITELN-state-ment voegt zelf weer een CR/LF toe. De laatste regel van de file wordt met een WRITE weggeschreven als deze niet met een CR/LF eindigt. Eindigt de laatste regel daar wel mee, dan is deze regel al door het filter weggeschreven. Als u met dit programma een Pascalsource bekijkt, zult u zien dat aan het eind van de tekst nog extra karakters in de file staan. Deze karakters zult u niet in de Pascaleditor zien, maar u wordt er juist iedere keer op gewezen als u een programma aan onze geachte hoofdredacteur aflevert. Zijn tekstverwerkertje is niet slim genoeg om dat er uit te filteren. (sorry Frank)

Zou u een programma maken dat alleen het aantal karakters in een file telt, dan zou het eenvoudiger zijn om een letter te lezen en direct daarna weer weg te schrijven. Hier is echter alvast een waarschuwing op zijn plaats. Het om en om inlezen en wegschrijven van een karakter is erg traag. In zo'n geval kunt u beter even een grote buffer nemen, en deze eerst vullen. Is de buffer helemaal gevuld, dan schrijft u deze in zijn geheel weer naar de file. Deze methode komt de snelheid zeer ten goede. Let u er bovendien op, dat u na afloop van het redirecten de pointers weer terug zet, anders gaat alles wat u op het scherm aan informatie krijgt ook rechtstreeks naar de nieuwe file, ook de foutmeldingen!

Variabelen

Zoals ik in het begin van dit artikel al vertelde zijn er ook nog wat variabelen. Waarschijnlijk was men bij Borland van mening dat deze niet nuttig te gebruiken zijn en hebben ze daarom deze variabelen niet in de handleiding uitgelegd. De variabele BufLen is een byte die bij initialiseren de waarde 126 krijgt. ledere verdere informatie ontbreekt en als er iemand is die mij kan vertellen wat ik met deze variabelen kan doen zou ik hem/haar zeer erkentelijk zijn. De variabele CBreak is nog veel fraaier. Deze staat in het geheel niet in de handleiding en is verder in geen enkel boek terug te vinden. Het is een Boolean die aangeeft of de compiler directive C wel of niet actief is. Door boven in uw code op te nemen { $c+ } of { $c- } kunt u aangeven of er tijdens het programma gecontroleerd moet worden op CTRL-S en CTRL-C. Deze compiler directive is globaal voor een volledig programma. U kunt dus niet halverwege de code deze check weer uitzetten. Met behulp van de variabele CBreak kan dat wel. U moet dan als compiler directive opnemen { $c+ } en kunt dan binnen uw programma de controle uitzetten door

CBreak: =FALSE. Overigens kunt u de { $c+} ook weglaten omdat deze als begin waarde al aan staat.

Ik hoop dat u met de informatie in dit artikel enigszins overweg zult kunnen. Het onderwerp pipelining en redirec-tion is niet iets waar de gemiddelde MSX'er vaak mee te maken heeft, ook ik niet, en het is dan ook lastig om een programma te schrijven zonder de fijne kneepjes te kennen. Ik heb het voorbeeld op de disk dan ook geschreven zonder gehinderd te worden door enige voorkennis.