Action deel 3

Zoals beloofd in het vorige artikel, gaan we deze keer wat verder in op het For Next statement.

Het For Next statement gebruikt u om de computer te vertellen een bepaalde taak een X aantal keren uit te voeren.Een simpel Basic voorbeeld:

```
10 FOR X=1 TO 5
20 A=X
30 PRINT A
40 NEXT X
```

Als u dit programma runt worden de volgende waarden op het scherm geprint : 1 2 3 4 5. Als X gelijk is aan 5 dan wordt de For Next loop verlaten. We zeggen dan, aan de voorwaarde dat X is gelijk aan 5 is voldaan.
In Basic is dit de enige manier om het For Next Statement te vertellen, hoeveel keer het een bepaalde taak moet doen. Bij Action! ligt dit anders. Een opsomming van de mogelijkheden.

```
for x=1 to 10 do ;**** ; Wordt als REM gebruikt od ----------
while x<10 do x==+1 od ----------
d   ==+ until x>10 od
```

Eerst wat uitleg

Nu ziet dit er allemaal verwarrend uit, daarom eerst wat uitleg.In Action! is het Basic For Next statement gelijk aan het Action! statement do od.Beide zijn immers een "LOOP" waarbij de opdrachten die in de loop staan worden uitgevoerd. Het Action! statement For Next wordt gebruikt om aan do od door te geven hoeveel keer de loop moet worden door lopen.
Het Action! statement while wordt gebruikt om de loop uit te voeren zolang aan een bepaalde voorwaarde wordt voldaan.In dit geval is dat:is X kleiner dan 10?Zo ja, voer dan de loop uit.Zo nee, verlaat dan de loop.
Het Action! statement until wordt gebruikt om de loop uit te voeren tot er aan een bepaalde voorwaarde wordt voldaan.In dit geval is dat:is X groter dan 10?Zo nee, voer de loop uit.Zo ja, verlaat de loop.
Ik begrijp dat dit pittige kost is, daarom nog enige voorbeelden.

Het For Next statement:

```
procedure voorbeeld()
Byte x,a;eerst variabelen declareren. for x=1 to 5 do a=x ;a is nu gelijk aan x printbe(a) ;printen van een Byte variabele od ;einde van de loop return ;einde van de procedure
```

Het While statement:

```
procedure voorbeeld()
byte key=765 key=255 while key<>33 do printbe(key) od return
```

Hier is de variabele key gelijk gemaakt aan het toetsenbord register.Als u nu op een toets drukt wordt de (Hardware) waarde van deze toets in key opgeborgen.Zolang key niet gelijk is aan 33 (Spatiebalk) wordt de loop uit gevoerd.Handig om iets van een bepaalde toets afhankelijk te maken.

Het Until statement:

```
procedure voorbeeld()
byte key=765 key=255 do printe("Hallo dit is een voorbeeld") printe("Druk op de spatiebalk om te stoppen.") until key=33 od return
```

Dit lijkt een beetje op het while statement,maar heeft toch een eigen plaats binnen Action! Pascal en C programmeurs zullen een en ander wel herkend hebben in de bovenstaande procedures.
Zo, ik hoop dat u ondertussen niet in slaap bent gevallen en het allemaal nog een beetje begrijpt.Mocht u vragen en/of op- of aanmerkingen hebben dan kunt u mij schrijven via het volgende adres:
Theo Okhuijsen t.a.v. Action! AM Postbus 18773 2502 ET Den-Haag
De demo diskette kunt u bestellen door 12,50 gld. over te maken op giro 3836186 t.n.v Theo Okhuijsen te Den-Haag.

Happy Bytes, Theo Okhuijsen, Den-Haag.

P.S. Mensen die met Action! willen beginnen kunnen de cartridge bestellen bij Telekoder (010-4133495).

Basicode en Atari-Basic

Al weer enige tijd is voor de 800 XL en de 130 XE soft- en hardware beschikbaar voor het inlezen en vertalen van Basicode programma's. Er blijven echter na vertaling nog wat kleinere problemen over (zie de handleiding bij het Basicode interface).

Over een aantal van deze problemen kreeg ik regelmatig vragen tijdens de regionale avonden in Dordrecht. De meeste vragen gingen over *stringbewerkingen.* Ik zal trachten een en ander te verduidelijken en mogelijke oplossingen aan te dragen. Eerst echter een beetje theorie: Een string is een rij karakters die door de computer als 1 variabele behandeld kan worden. Bijvoorbeeld A$="reeks". De variabele A$ is dan een reeks van vijf karakters. Met Atari-Basic kun je een hele serie bewerkingen met strings verrichten, waarvan ik aanneem dat u ze beheerst. Basicode echter gebruikt ook andere commando's, te weten: LEFT$(A$,3) betekent : neem van A$ de eerste drie karakters Right$(A$,3) betekent: neem van A$ de laatste drie karakters.MID$(A$,3,3) betekent: neem uit A$ drie karakters te beginnen bij het derde. Dus in dit geval de karakters 3, 4 en 5.
Om met Atari-Basic hetzelfde resultaat te krijgen moet enig programmeerwerk verzet worden, en wel als volgt: LEFT$(A$,P) vervangen door: A$(1,P).RIGHT$(A$,P) vervangen door: A$(LEN(A$)-P,LEN(A$)) , MID$(A$,P,Q) vervangen door: A$(P,P+Q-1)
Een volgend probleem was het 'optellen' van strings, bijvoorbeeld C$=A$+B$. Dit valt op te lossen op de volgende manier: C$=A$:C$(LEN(C$)+1)=B$. Let er op dat bij Atari-Basic alle strings vooraf geDIMensioneerd moeten worden!
In Basicode is ook toegestaan: PRINT TAB(K) A$. Dit vraagt om iets meer bewerking: REGEL=PEEK(84): KOLOM=PEEK(85): POS.KOLOM+K,REGEL: PRINT A$
Ik hoop dat ik hiermee een groot aantal vragen en vraagjes heb beantwoord. Veel succes bij het aanpassen van Basicode programma's.
Bovenstaande geldt ook voor het eventueel aanpassen van Basic programma's voor computers die zich van een versie van Microsoft Basic bedienen. Let wel: PEEK en POKE opdrachten zijn nooit direkt uitwisselbaar. Er zijn nog meer verschillen tussen de Microsoft Basics en Atari-Basic. Als er belangstelling voor is wil ik daaraan in de toekomst ook aandacht besteden. Maar dan liefst aan de hand van concrete vragen.

C.Baarda, Neptunusstraat 34, 3331 SK Zwijndrecht.

Vervolg van blz. 39

vuurknop te drukken en met de stick kun je remmen , gasgeven of op je achterwiel gaan rijden om je sprong beter te kunnen uitmikken. Je kunt zelf uit een negental verschillende parcoursen kiezen. Bijzonder is nog, dat het scherm in twee windows is verdeeld, zodat je met twee spelers tegen elkaar kunt racen. De reviewer van Atari User was bijzonder enthousiast over dit spel en gaf het heel hoge waarderingen voor graphics, speelbaarheid en "waar voor je geld". Voor de nabije toekomst zijn al weer drie nieuwe spellen aangekondigd, namelijk Vegas Poker, een poker-simulatie, verder Bump, Set, Spike, naar ik meen een snooker-spel en tenslotte Spellbound. Vooral naar dat laatste spel kijk ik uit, omdat het op de Spectrum werkelijk een zeer fraai spel is, een adventure maar dan zonder het voor velen zenuwslopende zoeken naar het juiste woord. Op het scherm staat namelijk een aantal commando's, die je met je joystick kunt ingeven als je mannetje op een kritiek punt is gekomen. De Spectrum versie heeft zeer lovende kritieken gekregen en ik ben heel benieuwd, wat de Mastertronics programmeurs uit de veel grotere mogelijkheden van de Atari weten te halen. De exacte levertijd van deze spullen is nog niet bekend, maar Mastertronics heeft de laatste maanden met de regelmaat van een klok iedere maand een nieuw spel uitgebracht, dus er is alle reden te hopen, dat ook de nog niet verschenen zaken binnen korte tijd op de markt zullen zijn. Micro City, de Nederlandse importeur heeft ze in ieder geval al in zijn catalogus opgenomen, dus daar is men zeker van zijn zaak. Een heel andere manier van Budget Games tenslotte zijn de zgn. compilaties, meer spellen op een tape of disk en dan voor de prijs van een spel. Zo bijv. de Smash Hits I-IV van English Software. Ook hiervoor geldt: veel geboden voor weinig geld. Het in de S.A.G. softwarewinkel nog maar in enkele exemplaren voorradige Smash Hits IV (alleen nog op tape, de disk is al uitverkocht) komt zo op nog geen 5 gulden per spel. Voor ons Atarianen alles bij elkaar een zeer vreugdevolle zaak en een uitstekende gelegenheid om te laten zien, dat er wel degelijk een markt is voor goede en betaalbare software. Op naar Uw leverancier dus!!!

Wim Denie

Action deel 4

Welkom bij het vierde deel van de Action! rubriek. Nadat we in deel 2 iets over variabelen en in deel 3 iets over het For Next statement hebben geleerd, gaan we nu een Action! programma bekijken. Voor het gemak is de listing voorzien van regelnummers. Normaliter horen deze nummers niet thuis in Action!. Let hier dus op!

In regel 18 worden de Variabelen Ab en Key gedeclareerd. Deze zijn van het Type Byte. Ab wordt gebruikt als een teller en Key wijst naar het toetsenbord register. Als u nu op een toets drukt, wordt automatisch de (Hardware) waarde van deze toets in geheugen locatie 764 gezet. Key wijst als het ware naar deze locatie. Handig om op een snelle manier te zien, of er op een toets is gedrukt.
In regel 20 wordt de Variabele Ac gedeclareerd. Deze is van het Type Card en wordt als teller gebruikt. Daar deze teller gr r wordt dan 255 is het nodig om een card te gebruiken.
In regel 22 wordt de procedure Main () gedeclareerd. In dit geval is dit de enige procedure, maar een Action! programma kan uit meer Procedures bestaan. In regel 24 wordt een Graphics 8 scherm geopend. Ziet u dat Action! hier erg veel op (Atari) Basic lijkt. Handig!
In regel 26 worden de schermkleuren ingesteld met behulp van het Setcolor statement.

Nu regel 28. De variabele Color is een interne Action! variabele en deze naam mag door u niet gebruikt worden voor andere variabelen. Color geeft aan met welke kleur er getekend moet worden. Key wordt hier op 255 (geen toets gedrukt) gezet.
In de regels 30 t/m 48 wordt een raster patroon op het scherm getekend door twee For Next loops. De eeste loop neemt de horizontale lijnen voor zijn rekening en de tweede loop de verticale lijnen. Weer lijken de statements Plot en Drawto erg veel op hun Basic broertjes.
In regel 50 wordt opdracht gegeven om de volgende loop net zolang uit te voeren, totdat er op een toets wordt gedrukt. Regel 52 tenslotte bevat het Return statement dat het einde van deze procedure aangeeft.

Hoe is nu een Action! programma opgebouwd? Eerst declareert u de te gebruiken Variabelen. Daarna declareert u de Procedure die u wilt laten uitvoeren. dan volgen de Statements binnen de Procedure. Tenslotte het Return statement dat de Procedure afsluit.
Maar als ik nu meer Procedures heb? Dan wordt de laatst gedeclareerde Procedure het eerst uitgevoerd en zult u de andere Procedures van uit deze Procedure moeten oproepen. Bekijk hiervoor Listing 2 om te zien hoe dat er uit ziet.
Hoe compileer ik nu zo'n listing? U tikt de listing met behulp van de Action! Editor in (regelnummers weglaten!) en controleert of u fouten hebt gemaakt. Druk nu op Control, Shift en M tegelijk en u komt in de Action! Monitor terecht.

Tik nu C en druk dan op de return toets. Uw programma wordt nu gecompileerd. Als het klaar is, drukt u op R en dan op de return toets. Uw programma gaat nu (hopelijk) draaien en u krijgt het fraaie raster patroon te zien. Als u hier genoeg van heeft, drukt u op een toets en het programma stopt dan.

Vanaf nu zal ik elke keer een fout behandelen, die de beginnen Action! programmeur kan overkomen. Bekijk het volgende statement: For x=1 to 255. Als x nu een variabele van het Type Byte is, wat is hier dan fout? Uit leg volgt. Action! voert de For Next loops uit, door de waarde van de teller met een te verhogen, om deze vervolgens te vergelijken met de eind waarde. Is de teller hoger dan de eind waarde, dan wordt de loop verlaten. Als x nu 255 bedraagt, dan x met een verhoogt. Er treedt nu overflow op, dat wil zeggen dat x nul wordt. dit komt door de manier warop de microprocessor de gegevens manipuleert. X kan nu dus nooit groter dan 255 worden, er ontstaat dus een eindeloze loop. Dit zelfde verhaal geldt ook voor de Card en de Integer variabelen. let er dus altijd bij For Next loops op, dat de gewenste eindwaarde kleiner is dan de grootst toegestane waarde voor het Typ[e variabele van de gebruikte teller. Ik begrijp dat dit pittige kost is, maar later zal het u veel zorgen besparen. dit was het weer voor deze keer. In deel 5 vertel ik iets over de opbouw van de cartridge. De demo diskette is te bestellen door 15 gld over te maken op giro 3836186 t.n.v. Theo Okhuijsn te Den Haag.

Happy Bytes, Theo Okhuijsen

```
10 ;Action! Listing 1.
12 ;door Theo Okhuijsen.
14 ;(c) 1986 by T.O.S.S.
16 ;----------------------
18 Byte Ab,Key=764
20 Card Ac
22 Proc Main()
24 Graphics(8+16)
26 Setcolor(1,0,0) Setcolor(2,0,14)
28 Color=1 Key=255
30 For Ac=0 to 319 Step 3
32  do
34   Plot(Ac,0)
36   Drawto(Ac,191)
38  od
40 For Ab=0 to 191 Step 3
42  do
44   Plot(0,Ab)
46   Drawto(319,Ab)
48  od
50 While Key=255 do od
52 Return
```

```
0 ;Action! Listing 2
2 ;door Theo Okhuijsen.
4 ;(c) 1986 by T.O.S.S.
6 ;----------------------
8 Byte Ab,Key=764
10 Card Ac
12
14 Proc Initscherm()
16       Graphics(8+16)
18       Setcolor(1,0,0)
20       Setcolor(2,0,14)
22 Return
24
26 Proc Initcolor()
28       Color=1
30 Return
32
34 Proc Vertical()
36       For Ac=0 to 319 Step 3
38        do
40         Plot(Ac,0)
42         Drawto(Ac,191)
44        od
46 Return
48
50 Proc Horizontal()
52       For Ab=0 to 191 Step 3
54        do
56         Plot(0,Ab)
58         Drawto(319,Ab)
60        od
62 Return
64
66 Proc Toets()
68       Key=255
70       While Key=255 do od
72 Return
74
76 Proc Main()
78
80       Initscherm()
82       Initcolor()
84       Vertical()
86       Horizontal()
88       Toets()
90 Return
92
```

Action! deel 5

Zoals beloofd in deel 4 volgt nu een bespreking van de Action! cartridge, dit naar aanleiding van vele verzoeken. Daarom deze keer geen listing.

De Action! cartridge bestaat uit de volgende onderdelen: De Editor, de Monitor, de Compiler en de Library.
Met de Editor typt u het Action! programma in. In feite is deze Editor een volwaardige tekstverwerker. Een kleine opsomming van de mogelijkheden: Het opvragen van een directory van disk via een "window". Het opzoeken en vervangen van stukken tekst. Het verplaatsen en kopieren van tekst. De mogelijkheid om twee verschillende sources (bron-code) tegelijk te bewerken.
Met de Monitor kunt u het gecompileerde programma laten runnen. Via de Monitor kunt u het gecompileerde programma naar disk schrijven om het voor later te bewaren. Verder kunt u een aantal opties instellen die van belang zijn voor het runnen van uw programma. Zo kunt u het programma laten "tracen", alleen in graphics 0, u krijgt dan constant in beeld met welke procedure het programma bezig is.

Compiler

Met de Compiler compileert u de source (bron-code) die via de Editor is aangemaakt. Eigenlijk is de Compiler een onderdeel van de Monitor, maar wordt vanwege de omvang toch als een op zich staand deel beschouwd.
Ten slotte is er de Library. Dit betekent Bibliotheek en dat is het ook. Het is een verzameling van kant en klare routines die u op kunt roepen via de cartridge. Zo zitten er naast de Action! routines ook dingen in als: graphics, plot, drawto, sound, print enz. Handig en makkelijk!
De cartridge is 16K groot, maar door een truc wordt maar 8K geheugen van uw computer gebruikt.
Zie dat de hele gang van zaken rond een Action! programma sterk op het schrijven van een C programma lijkt? Eerst moet u een source aanmaken, die u daarna compileert. Hierna wordt het gecompileerde file automatisch gelinkt met de routines uit de Library. In C moet dit via een aparte, tijdrovende, stap gedaan worden.

Het volgende is voor de wat gevorderde Action! programmeur.Met de cartridge is het mogelijk om een file te maken dat ongeveer 30K aan object code oplevert. Weliswaar moet u dan van disk compileren, maar de mogelijkheid is wel aanwezig. Overigens kunt ook van de 130XE Ramdisk compileren. Dit versnelt het hele proces aanzienlijk. Het programma TOSS-PRINT voor de Atari printers is op deze manier gemaakt. Het programma neemt samen met alle buffers ongeveer 47K in beslag. U ziet dus dat professionele toepassingen op een eenvoudige en snelle wijze zijn te maken. De ontwikkelingstijd van dit programma was 4 maanden. Had ik hetzelfde programma in machinetaal moeten schrijven, dan schat ik dat het een jaar had geduurd. Afgezien van het feit, dat ik geen machinetaal beheers.
Ik hoop dat nu een en ander duidelijk is rond de opbouw van de cartridge. De volgende keer gaan we verder met de taal Action! en bespreken we een aantal routines uit de Libary. De demo-disk is te bestellen door 15 gulden over te maken op giro 3836186 van Okhuijsen te Den-Haag.

Happy Bytes, Theo Okhuijsen

Foutmeldingen en variabelen

Pilot, deel 3

Beste Pilotvrienden,
Eerst moet ik even een aantal dingen rechtzetten. Bij het samenstellen van het vorige nummer zijn in het Pilotverhaal een paar fouten geslopen, die tot misverstanden hebben geleid. Fouten die ik hierbij even recht moet zetten.

In het eerste voorbeeld werd in de laatste regel alleen "Harlingen" afgedrukt. Dat moet natuurlijk "uit Harlingen" zijn. Zo staat het ook in de listing. Ook zal de som uit het voorbeeld nooit zonder opdracht achter de vorige zin worden afgedrukt. In het volgende voorbeeld was het wel de bedoeling om de zaken achter elkaar af te drukken. Daarvoor hoorde er achter de zin een omgekeerde deelstreep (\). Als je een zin daarmee afsluit, wordt de volgende zin erachter geschreven.
Zo, ik hoop dat de problemen nu zijn opgelost. We gaan ons deze keer bezig houden met variabelen. Hoe kunnen we een antwoord opslaan om het later weer te kunnen gebruiken? We moeten dan eerst onderscheid maken tussen twee soorten variabelen. Elk heeft zijn eigen notatie. Voor beiden geldt dat ze gedurende het hele programma bewaard blijven. In Pilot kennen we:
1. Getalvariabelen:
- ze worden geschreven met een #
- daarachter mag maar 1 letter
- #A, #B, #C zijn dus goed
- #CIJFER, #1, zijn dus fout
- met getalvariabelen kun je alle bekende rekenkundige bewerkingen uitvoeren.

2. Stringvariabelen:
- ze worden geschreven met een $
- daarachter mogen letters en cijfers worden geschreven
- $NAAM, $NAAM2 zijn dus goed
- $..., $.N. zijn dus fout.

Nog een ding. Als je in Pilot wilt rekenen, moet je de zin beginnen met C: De computer weet dan dat het om rekenen (Calculate) gaat.
Kijk maar eens naar dit voorbeeld.

```
5 *DEMO
10 T:Hallo, hoe heet je?
20 A:$NAAM
30 T:Dag $NAAM, leuk je te zien.
40 T:Hoe oud ben je, $NAAM ?
50 A:#L
60 R:nu gaan we rekenen
70 C:#I=65-#L
80 T:Weet je $NAAM, dat je over #I
     jaar al met pensioen gaat?
90 E:
```

We zullen het programma nu eens gaan bekijken.
- Regel 5 geeft het programma een naam. De naam begint altijd met een ster.
- Regel 10-50 zijn wel bekend. Er wordt tekst afgedrukt en om invoer gevraagd. Je ziet dat je variabelen zomaar in de zin mag afdrukken.
- Regel 60 is nieuw. De R: staat voor REMARK. Een opmerking dus die niet wordt afgedrukt. Alleen voor jezelf.
- Regel 70 is de rekenregel. Hij maakt van #I een nieuwe variabele door de oude variabele #L van 65 af te trekken.
- Regel 80 mag nu ook bekend zijn.
- Regel 90 sluit alles af. De E: wordt als einde herkend.

Beginnen met een naam (*DEMO) en afsluiten met E: is erg belangrijk. Door een begin en eind kun je programma's later als procedures of subroutines in grotere programma's aanroepen. Dat zullen we in een volgende aflevering wel merken.
Nog twee tips voor mensen die boeken of informatie zoeken over Pilot:
- De oude Atari Connection had vaak een rubriek over Pilot. Hierin kun je veel informatie vinden.
- Het boek van David D. Thornburg, "Picture This", dat 200 pagina's bruikbare Pilotinformatie biedt, ligt in een aantal boekwinkels in de uitverkoop. Het kost dan nog geen tientje en is zeker de moeite waard.

Welnu, dat was het voor deze keer. Veel succes.

Peter Sikma

Theo Okhuijsen over Functions

Action!, deel zes

Welkom bij de Action! rubriek. In dit deel gaan we in op wat Functions zijn en wat we ermee kunnen doen. Een *Function* is bijna hetzelfde als een *Procedure* met dit verschil, dat een Function een waarde (getal) kan teruggeven en een Procedure dit nooit kan. Een simpel voorbeeld :

Voorbeeld van een Procedure :

```
Proc test()
byte a
a=10
return
```

Voorbeeld van een Function :

```
Byte Func test()
byte a
a=10
return(a)
```

Bij de Function wordt een waarde teruggegeven. In dit geval is dat de variabele a die de waarde 10 heeft. Als u een Function gebruikt, moet u de Function declareren, dat wil zeggen dat u van tevoren opgeeft, wat voor waarde de Function zal teruggeven. Dat mag een Byte waarde (0 t/m 255), een Card waarde (0 t/m 65535) of een Int waarde (-32768 t/m 32767) zijn. U spreekt van een Function, maar schrijft in een programma altijd het woord Func.

Waar gebruik ik een Function voor? Functions kunt u gebruiken voor het uit laten voeren van berekeningen, voor het opvragen en invoeren van gegevens en eigenlijk voor alles waar om een waarde wordt gevraagd.

Hoe gebruik ik een Function? Een Function gebruikt u op de volgende manier:
a=Inputb()

Er staat als het ware: voer de Function Inputb() uit en geef de uitkomst door aan a. a is een variabele en Inputb() is een eerder gedeclareerde Function. U moet er goed op letten dat a van het juiste type variabele is en dat de Function de juiste waarde doorgeeft aan a. U kunt bijvoorbeeld niet een Card waarde aan een Byte variabele doorgeven. In de Action! listing wordt een goed voorbeeld gegeven van wat een Function is, hoe je hem gebruikt en wat je ermee kunt doen.

Als u vragen heeft over de listing of iets niet begrijpt, of wat wilt weten over Action! schrijf dan naar:

Theo Okhuijsen
Action! Atari Magazine, Postbus 18773
2502 ET Den Haag
en sluit een gefrankeerde, aan uzelf geadresseerde envelop bij. Ik zal dan proberen om uw vraag zo goed mogelijk te beantwoorden. De volgende keer gaan we het hebben over pointers.

Happy Bytes,
Theo Okhuijsen

P.S. Wat mededelingen van huishoudelijke aard:

De Action! Demo-disk is te bestellen door vijftien gulden over te maken op giro 3836186 t.n.v. Theo Okhuijsen te Den Haag onder vermelding van Action! Demo.
Verder heb ik uit Amerika een disk boordevol met leuke Action! programma's (PD) binnen gekregen. Ook deze disk kunt u bestellen door vijftien gulden over te maken op het bovenstaande gironummer onder vermelding van Action! PD-disk.

```
;Action listing (Action deel 6)
;(c) 1987 Theo Okhuijsen
;voor Atari Magazine

Byte a,b,key=764,teller=540
Card ac,bc
;$$$$
Byte Func Randon()
Byte al
al=peek(53770);random register
return(al) ;geef getal terug
;$$$$
Card Func Bigrand()
Card al,bl
al=peek(53770);random register
bl=peek(53770);idem
al=al%bl ;verenigvuldig al en bl
return(al) ;geef uitkomst terug
;$$$$

Proc Maakschera()
graphics(0)    ;tekst schera
setcolor(1,0,0) ;helderheid tekens
setcolor(2,0,14);kleur schera
setcolor(4,3,0) ;kleur kader
position(6,0)
printe("Demonstratie van FUNCTIONS")
position(9,1)
printe("Door Theo Okhuijsen")
position(2,4)
printe("Druk op K voor een random getal <255.")
position(2,6)
printe("Druk op G voor een random getal >255.")
position(2,8)
printe("Druk op S voor stoppen van programma.")
return
;$$$$
Proc Main()
Card getal

do
 Maakschera()
 getal=0
 key=255 ;reset toetsenbord register
 while key=255 do od ;loopje
 if key= 5 then getal=randon()  fi ;K
 if key=61 then getal=bigrand() fi ;G
 if key=62 then exit            fi ;S
 position(2,11)
 print("Het random getal is : ")
 printce(getal) ;print het getal
 teller=100
 while teller>0 do od
od
return
;teller is een software counter die
;elke 50e seconde met een (1) wordt
;verminderd.key is het toetsenbord
;register
```

Action!, deel 7: de Pointer-brothers

In de zevende aflevering van de serie over de taal Action! behandelt Theo Okhuijsen het begrip 'pointers'. Volgens hem de beste vrienden die je in Action! kunt hebben.

Naast de fundamentele data types BYTE, CARD en INT kent Action! de zogenaamde 'extended' data types. Het data type Pointer behoort tot deze extended data types. Een pointer is een variabele waarvan we het adres waar die variabele naar wijst (point!) kunnen veranderen.

Hoe gebruik je een pointer? Als we in de listing de procedure INKEYS bekijken, zien we dat deze een tegenhanger heeft in de procedure INKEYSP. In INKEYS wordt de BYTE KEY gebruikt die naar het toetsenbordreister wijst, zodat we de inhoud van dit register kunnen uitlezen. Waar KEY naar wijst hebben we in het begin van het programma vastgelegd (gedeclareerd) en dit kunnen we niet meer veranderen. In de procedure INKEYSP gebruiken we in plaats van een BYTE variabele de pointer BPON om naar het toetsenbordregister te wijzen. We moeten deze pointer eerst vertellen waar hij naar moet wijzen. Dit doen we met het statement bpon=@key. Aangezien het adres waar KEY naar wijst 764 is, wijst ook BPON nu naar 764. We hadden ook het volgende mogen gebruiken: bpon=764.

```
;####

Proc Verschil()
;Verschil in TYPE pointer
put(125);Wis scherm
printe("Het verschil in DATA TYPE.")
printe("")
test=65025  bpon=@test
r   =@test  ipon=@test
print("TEST = CARD= ") printce(test)
print("BPON = BYTE= ") printbe(bpon^)
print("CPON = CARD= ") printce(cpon^)
print("IPON = INT = ") printie(ipon^)
printe("")
print("Adres TEST= ") printce(@test)
print("Adres BPON= ") printce(@bpon)
print("Adres CPON= ") printce(@cpon)
print("Adres IPON= ") printce(@ipon)
verder()

return

;####

proc main()

     maakscherm()
     verschil()
     printe("Einde demonstratie.")
     dummy=inkeysp()
return
```

Inhoud veranderen

Vervolgens kunnen we *de inhoud van het geheugen waar de pointer naar wijst,* veranderen. Dit gebeurt met het statement: bpon^=255. De inhoud van BPON, dat is locatie 764, bevat nu de waarde van 255. Het feit dat we een pointer naar een willekeurige geheugenplaats kunnen laten wijzen is reuze makkelijk en bespaart een hoop geheugenruimte. Immers, in plaats van een groot aantal variabeen kunnen we volstaan met één pointer die we naar het gewenste adres laten wijzen. Let er op dat de pointers zijn onderverdeeld in de fundamentele data types. Dus een BYTE pointer verwerkt een getal van 0 t/m 255, een CARD pointer verwerkt een getal van 0 t/m 65535 en een INT pointer een getal van min 32768 t/m plus 32767. Een goed voorbeeld hiervan geeft de procedure 'Verschil'. In deze procedure wijzen de pointers BPON, CPON en IPON naar de CARD variabele TEST, die de waarde 65025 heeft. De BYTE pointer BPON en de INT pointer IPON geven een totaal andere uitkomst dan de CARD pointer CPON.

Overzicht:
1. Pointer declareren
bpon=@variabele
bpon=764
2. Veranderen van inhoud van het adres waar pointer naar wijst
bpon^=255
bpon^=variabele

3. Rekenkundige bewerking met pointer
byte a
a=10*bpon^

Het @-teken (apestaartje) betekent het adres van een variabele. Het ^-teken (dakje) betekent de inhoud van het geheugen waar de pointer naar wijst. Het adres waar een pointer naar wijst is altijd een CARD groot! Het apestaartje is het teken dat je met Shift-8 krijgt en het dakje zit op de toets met het rechter cursorpijltje.
Hè hè, klaar. Ik besef dat een en ander pittige kost is, maar als je het artikel en de listing goed leest zul je zien dat pointers de beste vrienden zijn die je in Action! kunt hebben.

Halve zolen

Dan nu even iets heel anders. Steeds vaker krijg ik klachten van mensen die problemen hebben met het compileren van de Action! listings. Bij navraag blijkt in 99% van de gevallen dat deze mensen de **gekraakte** Action! op disk hebben. (Nu is dit op zichzelf geen ramp, want ook met de diskversie is het mogelijk om de listings (mits niet groter dan ca. 10K) te compileren.) In de meeste gevallen blijkt dan ook nog dat deze luitjes een

```
;Action! listing (Action! deel 7)
;(c) 1987 Theo Okhuijsen
;voor Atari Magazine
;Onderwerp : Pointers

Byte pointer bpon
Card pointer cpon
Int  pointer ipon
Byte key=764,dummy
Card test

;####

Byte Func Inkeys()

key=255
while key=255
 do
  ;do loop tot key<>255
 od

return(key)

;####

Byte Func Inkeysp()

bpon=@key
bpon^=255
while bpon^=255 do od

return(bpon^)

;####

Proc Verder()

poke(752,1);cursor weg
printe("")
printe("Druk op een toets!")
dummy=inkeys()

return

;####

Proc Maakscherm()

graphics(0)
setcolor(1,0,12)
setcolor(2,0,0)
printe("POINTER demonstratie")
verder()

return
```

versie hebben waar de een of andere halve zool een poging heeft gedaan om de teksten in het Nederlands te vertalen en daarnaast ook nog op de gekste plaatsen kreten als 'Wrecked by the Amazing Fungus' heeft neergezet.
Werken met deze versie is dus vragen om moeilijkheden. Vaak slaat het systeem op hol en als je probeert een source file te saven wordt de disk geformatteerd. Zorg dus dat je de goede diskversie krijgt, of nog liever de cartridge. Dat werkt veel makkelijker en veiliger. De Computershop Utrecht heeft er nog een stel liggen, samen met de Action! Toolkit. Met de Toolkit kun je de programma's ook zonder de cartridge laten lopen. OK, en dan verder niet meer klagen over listings die niet deugen.

Heb je vragen of opmerkingen over dit artikel? Schrijf dan naar:
Theo Okhuijsen
Action! Atari Magazine
Postbus 18773
2502 ET Den Haag
en sluit een gefrankeerde en aan jezelf geadresseerde envelop bij. Ik zal mijn best doen om je vraag zo goed mogelijk te beantwoorden. De volgende keer gaan we het hebben over het gebruik van OPERATORS in Action!

Happy Bytes
Theo Okhuijsen
PS: De Action! demo-disk en de Action! PD-disk zijn te bestellen door voor ieder afzonderlijk vijftien gulden over te maken op giro 3836186 t.n.v. Theo Okhuijsen, Den Haag, o.v.v. Action! Demo en/of Action! PD.

De superpokes gekraakt

Iedere Basic-liefhebber kent ze wel, de POKE-opdrachten om een listing onleesbaar en onlistbaar te maken. Maar Laurens Koenen en Jules Bastiaans zijn erin geslaagd ze te omzeilen, m.b.v. de programma's The Scanalyzer en Speedscript.

Om een programma te beschermen duiken aan het eind altijd de volgende twee regels op:
32000 FOR I = PEEK(130) + 256 * PEEK(131) TO PEEK(132) + 256 * PEEK(133): POKE I,155: NEXT I
Hiermee worden alle variabelen in de listing vervangen door een return-teken. En dan komt de volgende:
32010 POKE PEEK(138) + 256 * PEEK(139) + 2,0: SAVE "D:PROGRAM": NEW
Deze regel heeft tot gevolg dat het programma alleen maar van disk kan worden gerund met RUN "D:PROGRAM".

Met behulp van wat software (Speedscript 3.0 en The Scanalyzer) is het mogelijk deze poke-opdrachten te omzeilen. Formatteer eerst een diskette op single density en zet daar een kopie van het te kraken Basic-programma op. Start nu het programma The Scanalyzer. Na het verschijnen van het menu kiest men de optie Basic-Lister. Hiermee kunnen beschermde Basic-programma's worden gelist naar het scherm, de printer of een diskfile. Nadat Basic-Lister is geladen doet men de disk met het Basic-programma in de drive. Tik één in voor het gevraagde drivenummer. Men kan vervolgens de directory van de disk bekijken. Op de vraag welke file 'gemaakt' moet worden typt men de naam van het Basic-programma in.

Allemaal spaties
Nu verschijnt de variabelentabel. Als de gevreesde poke-opdrachten zijn gebruikt staan er allemaal spaties. Op de vraag of men deze tabel wil gebruiken typt men 'N'. Het programma vervangt de stringvariabelen nu door S0$, S1$ enzovoorts, en de numerieke variabelen door N0, N1 etc. Als het hiermee klaar is moet men het ergens naar toe schrijven. Voor de eerste keer kan men het naar het scherm schrijven, maar het moet eigenlijk naar een diskfile. Optie C wordt dus gekozen en voor de filenaam neemt men bijvoorbeeld D:LKJB.

Haal de disk met het Basic-programma uit de drive en laad vervolgens Speedscript 3.0. Laad nu de file LKJB en bekijk 'm eens rustig in de tekstverwerker. Het hele Basic-programma is ingeladen en is nu zichtbaar, compleet met de vervangen variabelen. Aan het eind van de file staat een aantal foutmeldingen; zoek de regel waar de poke-opdrachten beginnen, verwijder deze regels en alles wat er onder staat (ook de foutmeldingen), en save de overgebleven file nu onder de naam LKJB.DUP.
Verwijder de disk en start opnieuw op met Basic en DOS. Typ nu ENTER "D:LKJB.DUP" en druk op return. Typ na het laden LIST in en hopla, daar is het beschermde Basic-pro gramma. Het is nu volledig listbaar en alles kan bekeken en veranderd worden. Nu kan men het normaal op disk saven. Tip: met Speedscript kunnen de variabelen een andere naam krijgen dan N0 of S0$, met de functies FIND (zoeken) en REPLACE (vervangen).

Bijna elk beschermd Basic-programma is op deze manier om te zetten naar een programma waaruit beginners de trucs en de technieken van de experts kunnen halen. En dat kan volgens ons voor iedereen alleen maar voordeel opleveren.

Veel succes ermee,
Laurens Koenen/Jules Bastiaans

Operatoren in Action 8

Welkom bij deel acht van de Action! rubriek. De laatste tijd is de rubriek onregelmatig verschenen door problemen met de verzending van de kopij. Een en ander is nu opgelost en we gaan met frisse moed door. In dit deel gaan we het hebben over operatoren.

Operator is een Engels woord dat zich in dit geval het beste laat vertalen met rekenkundige bewerking. Action! beschikt over een groot aantal operators die we in drie groepen kunnen verdelen. De echte rekenkundige, zoals optellen, vermenigvuldigen en aftrekken. Deze groep noemen we de arithmmetic operators. Verder zijn er operators die condities testen. Deze groep heet de relational operators. De derde groep manipuleert getallen op bit nivo. Deze groep noemen we de bit-wise operators. De eerste twee groepen behandelen we in dit deel en de bit-wise operators komen in het deel over het gebruik van machinetaal in Action! ter sprake.

Rekenkundige operators:

+	is optellen.
−	is aftrekken.
*	is vermenigvuldigen.
/	is delen.
MOD	is het restant van een deling.

Relationele operators:

=	test op gelijkheid.
<>	test op ongelijkheid.
>	test op groter dan.
>=	test op groter of gelijk aan.
<	test op kleiner dan.
<=	test op kleiner of gelijk aan.
AND	logische and.
Or	logische or.

Voor het assignment (gelijk maken aan) teken wordt = gebruikt, maar dat was al bekend. De listing geeft een voorbeeld hoe de diverse operators worden gebruikt.

Overzicht:
Voor rekenkundige bewerkingen gebruiken we de arithmetic operators. Voor het testen van condities de relationele operators. Voor manipuleren van getallen op bit niveau de bit-wise operators.

De volgende keer gaan we een programma schrijven dat alles wat we tot nu toe geleerd hebben in de praktijk brengt.

Happy bytes, Theo Okhuijsen

```
;Demo van Action! operators
;Action! deel 8
;(c) 1987 by Theo Okhuijsen

Byte a=9,b=2,key=764
;key is toetsenbord register

;****

Proc Arithmetic()

printe("Rekenkundige operators :")
pute() printe("A=9 B=2") pute()

print("A + B   = ") printbe(a+b)
print("A - B   = ") printbe(a-b)
print("A * B   = ") printbe(a*b)
print("A / B   = ") printbe(a/b)
print("A MOD B = ") printbe(A mod b)
pute()

return

;****

Proc Relational()

printe("Relationele operators :")
pute() printe("A=9 B=2") pute()

if a>b then
printe("A is   > B? Ja") fi

if a<>b then
printe("A is   # B? Ja") fi
;# is hetzelfde als <>

if b<a then
printe("B is   < A? Ja") fi


if a>5 or b>5 then
printe("A of B > 5? Ja") fi

if a>0 or b>0 then
printe("A en B > 0? Ja") fi
pute()

return

;****

Proc Main()
     graphics(0)
     arithmetic()
     relational()
     printe("Druk op een toets.")
     key=255 while key=255 do od

return
```

Nogmaals bytes, cards en integers

De Action! Rubriek

9

Welkom bij deel negen van de Action! rubriek. In dit deel leg ik op veler verzoek nog eens uit hoe het zit met de bytes, cards en integers. Zoals bekend moet je in Action! *elke* variabele die men in een programma wil gebruiken, aan het begin van het programma declareren. Dat houdt in dat je de variabele een naam geeft en zegt van welk type de variabele is.

Juist met dat *type* hebben veel mensen moeite. Wat past er in een byte en hoe groot mag een card zijn? Om het goed uit te leggen moet ik in het kort wat vertellen over de 6502 CPU (de centrale processor) die in ons computertje zit en over de taal die deze CPU spreekt.
De enige taal die de 6502 verstaat is 6502 machinetaal. Alle andere talen zoals Basic, Logo of Action! worden, elk op een andere manier, in 6502 machinetaal vertaald. De methode waarop Action! met getallen omgaat is dus de methode die de 6502 hanteert. De 6502 CPU verwerkt getallen die acht bits (één byte) groot zijn. Bit nul van een byte noemt men het 'least significant bit' (lb) en bit 7 heet het 'most significant bit' (mb). Elk bit in een byte stelt een bepaalde waarde voor die terug te vinden is in de tabel in figuur 1. Als je de waarden van alle bits van een byte bij elkaar optelt, krijg je als uitkomst 256. Een byte kan dus de getallen 0 t/m 255 bevatten. Wil je met een groter getal dan 255 rekenen, dan moet je een 'word' gebruiken dat uit twee bytes bestaat.

Cards en Integers

De eerste byte van een word heet het 'least significant byte' (LSB), het tweede byte is het 'most significant byte' (MSB). Het LSB bevat een getal waarbij het getal uit het MSB, vermenigvuldigd met 256, wordt opgeteld. De uitkomst van 255 + (255 * 256) is 65535 en dit is het grootste getal dat in een card past. Een card is in feite hetzelfde als een word!
Nu een integer, dat is wat moeilijker. Een integer kan zowel een positief als een negatief getal zijn. Omdat de 6502 niets afweet van plus of min, gebruikt men daarvoor een truukje: als bit 7 van het tweede byte 'geset' is, dan wordt het getal als negatief beschouwd en als het bit 'clear' is, dan is het getal positief. Doordat we een bit gebruiken als vlag (sign) beperken we de grootte van het getal dat in een integer past.

Eindeloze lus

We weten nu hoe de 6502 met getallen omgaat en kunnen ons wijden aan het

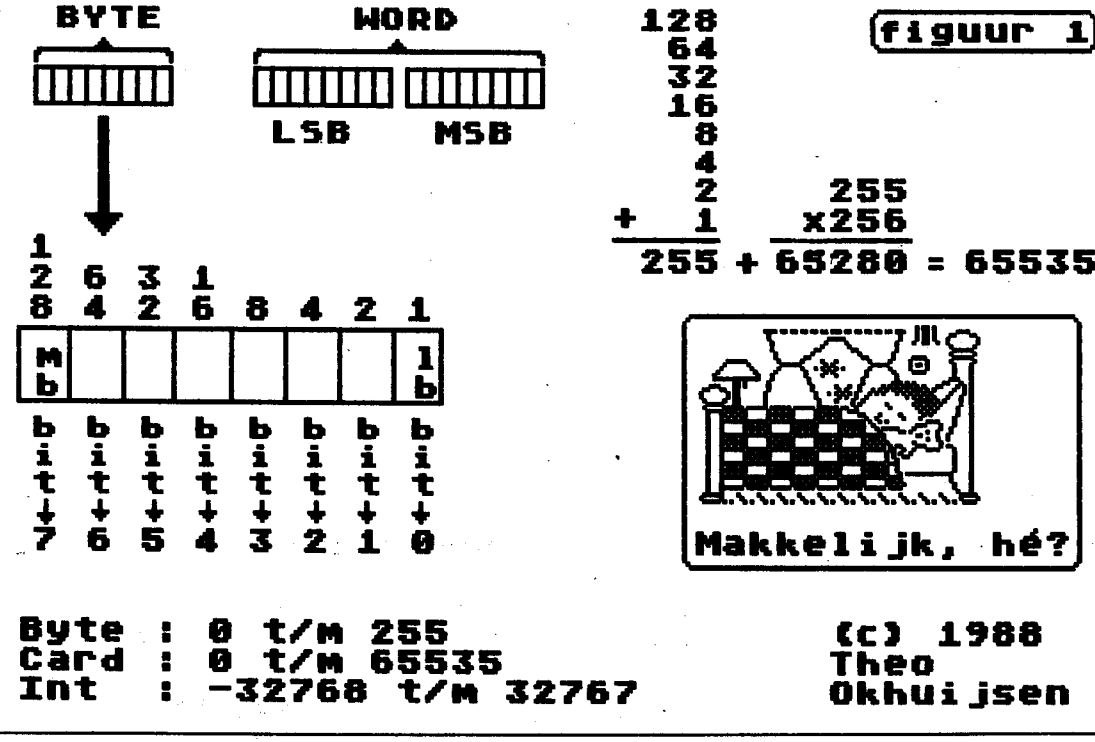
```
;Action! listing, deel negen
;Uitleg over TYPE variabelen
;(c) 1988 Theo Okhuijsen (25 mei 1988)
;Exclusief voor Atari Magazine

Byte A,B,Key=764
Card C
Int  I

;***

Proc Inkeys()

;Wacht tot een toets wordt gedrukt.

Pute()  ;lege regel
Printe("Druk op een toets.")
Key=255 While key=255 do od

Return

;***

Proc Setup()

;Wis scherm, stel kleuren in.

Graphics(0)
Setcolor(1,0,12)
Setcolor(2,0,0)
Print("Welkom bij deel 9 van de ")
printe("Action rubriek")
Inkeys()
Pute()

Return

;***

Proc Rekenen()

A=200 B=100 C=1 I=-1
```

volgende probleem(pje). Iedereen die wel eens het volgende heeft geprobeerd: FOR X = 0 TO 255 DO OD, weet dat deze lus niet netjes 256 keer wordt doorlopen maar zichzelf eindeloos herhaalt als X van het type byte is. Hoe komt dit? Als X van het type byte is, dan past 255 er toch in? Klopt als een bus. De oorzaak ligt elders. Action! test aan het einde van elke lus of deze verlaten moet worden. Daartoe wordt de variabele die de lus bestuurt met één (of de step size) verhoogd en vervolgens wordt getest of deze variabele groter is dan het opgegeven eindgetal. Als je nu een byte dat 255 als getal bevat met één verhoogt, is de uitkomst niet 256 maar nul! En omdat nul kleiner is dan 255 wordt de lus opnieuw doorlopen en nooit verlaten. Neem daarom als grootste eindgetal voor een variabele het grootste getal minus één dat toegestaan is voor dat type variabele.

Het was taaie kost deze keer, maar samen met het overzicht in figuur 1 en de Action! listing moet het duidelijk zijn. De volgende keer een herhaling over functions en procedures.

Happy Bytes, Theo Okhuijsen

```
;De uitkomst van bewerkingen met twee
;dezelfde TYPES variabelen, hoeft niet
;perse een uitkomst te geven van het
;gebruikte TYPE variabele .Hieronder
;een paar voorbeelden.


Print("A * B = ") Printce(A*B) ;CARD!
Print("A / B = ") Printbe(A/B) ;BYTE!
Print("C - B = ") Printie(C-B) ;INT!
Print("I + A = ") Printce(I+A) ;CARD!

;Let op het volgende voorbeeld!

Print("A - B = ") Printbe(A-B) ;BYTE
Print("A - B = ") Printce(A-B) ;CARD
Print("A - B = ") Printie(A-B) ;INT

;Zolang de uitkomst maar legaal is,
;voor dat TYPE variabele, mag dat TYPE
;gebruikt worden.

Pute()
Inkeys()

Return

;***

Proc Main()

Setup()
Rekenen()

Return
```



Programmeren in Action!, deel 9 10

Het IF-statement

Voor het beïnvloeden van de loop van een programma beschikt Action! over een aantal statements. Eén daarvan is het if-statement, dat Theo Okhuijsen deze keer behandelt.

Iedereen die in Basic programmeert zal vroeg of laat een regel schrijven als 10 IF A=1 THEN..., wat betekent dat als A de waarde 1 heeft, de instructie na THEN moet worden uitgevoerd. Action! beschikt ook over een if-statement, en dat w : op precies dezelfde manier als het if in (Atari) Basic. Alleen is het Action!-if wat uitgebreider.

Ik zal het uitleggen aan de hand van bijgaande listing. We beginnen bij de procedure *main()* omdat die als eerste wordt uitgevoerd. Main() begint via de procedure *init()* met het op nul zetten van een aantal variabelen die we zullen gebruiken om te testen. De volgende procedure is *enquete*. Daarin worden wat oubollige vragen gesteld om daar wat mee te gaan doen. Op de functie *scompare* zal ik in een volgend artikel uitgebreid terugkomen. Nu op naar *demo1()* waar het eerste if-statement wordt gebruikt. Demo1 kijkt of *lid* gelijk is aan nul en zo ja, dan wordt een hoerakreet op het scherm geprint. Lid zijn van de SAG is een fijne zaak, en dat mag iedereen weten. Demo1() zal verder geen problemen opleveren, hier wordt if in z'n meest elementaire vorm gebruikt: als iets zus is, doe dan zo.

Demo2() wordt al iets lastiger. Eerst wordt *goed* op nul getest en als dat zo is verschijnt weer een jubelkreet. Maar na de eerste test volgt niet *fi* ten teken dat de test afgelopen is, maar staat er *else*. Dat is Engels voor anders: als goed nul is, print dan 'fijn', print *anders* 'jammer'. Ook dit zal na enig oefenen geen problemen opleveren.
Als laatste procedure blijft demo3() over en hier begint het echte werk. Weer wordt een variabele op nul getest, maar in plaats van else staat er *elseif*. Elseif wordt gebruikt om een serie variabelen te testen. Zodra er een test is die waar is, wordt dat statement uitgevoerd en wordt de testlus verlaten. Probeert u maar eens verschillende antwoorden uit. Ter verduidelijking: in demo3() wordt altijd òf helemaal geen statement uitgevoerd (geen van de drie testen is waar) òf altijd maar één (deze zin was dus een voorbeeld van het if-statement).
Dit besluit de uitleg over het if-statement. De volgende keer gaan we een Action!-programma schrijven dat alles wat ik tot nu toe behandeld heb in praktijk brengt.

Happy Bytes, Theo Okhuijsen

```
;Action! listing, deel tien
;Uitleg over het if statement
;(c) 1988 Theo Okhuijsen (8 aug. 1988)
;Exclusief voor Atari Magazine

byte lid,goed,andere,key=764
byte array antwoord(255)
;***
proc demo1()
;geeft voorbeeld van het if statement

if lid=0 then
   printe("Hoera! U bent lid!")
fi

return
;***
proc demo2()
if goed=0 then
   printe("Fijn! Houden zo!")
   else          ;Let op!
   printe("Jammer! Helaas!")
fi
return
;***
proc demo3()
if andere=0 then
   printe("U houdt de Atari!")
   elseif goed=0 then
   printe("De Atari bevalt goed!")
   elseif lid=0 then
   printe("U houdt de Atari!")
fi
return
;***
proc enquete()

printe("Welkom bij de Sag Enquete.")
printe("Type JA of NEE en druk dan")
printe("op de return toets.")
printe("Bent u lid van de Sag?")
inputs(antwoord)
lid=scompare(antwoord,"JA")

printe("Bevalt de Atari goed?")
inputs(antwoord)
goed=scompare(antwoord,"JA")

printe("Wilt u een andere computer?")
inputs(antwoord)
andere=scompare(antwoord,"JA")
Return

;***
proc init()

lid=0 goed=0 andere=0

return
;***
proc main()

init()
enquete()
demo1()
demo2()
demo3()

return
```

Programmeren in Action! deel 11

Structuur in Action!

Dit artikel verschilt van de artikelen die tot nu toe verschenen zijn. Het gaat deze keer niet over de taal Action! zelf, maar hoe je een programma schrijft in Action! Ik geef een aantal tips en doe wat voorstellen om het programmeren in Action! een stuk makkelijker en overzichtelijker te maken. Maak je geen zorgen als er dingen in de listing staan die we nog niet behandeld hebben. Dat komt in een later artikel wel aan de orde. Sla eventueel de Action! handleiding er op na.

Hoe schrijf ik een programma in Action! is een vraag die iedereen die in Action! programmeert zichzelf stelt. Gewoon pats boem beginnen en maar op het toetsenbord intikken wat ons te binnen schiet? Of eerst het hele programma van A tot Z op papier zetten compleet met schema's en stroomdiagrammen? De eerste methode valt eigenlijk meteen af. Wij zijn niet in Action! gaan programmeren om op de houtje-touwtje manier te gaan zitten klooien. De structuur van de

```
 Het jaar 2000 (Uw leeftijd in 2000)
,Action! listing deel 11
;(c) 1988 Theo Okhuijsen
;9 september 1988, 15:10 pm
;Exclusief voor Atari Magazine

Byte key=764

Card leeftijd,resultaat

;***

Proc Setup()

Graphics(0)
Setcolor(1,0,14)
Setcolor(2,0,0)
Printe("Welkom in het jaar 2000")
Pute()
print("Hoe oud was U in 1988?")

Return

;***

 ard Func vraag()

Leeftijd=Inputc()

Return(leeftijd)

;***

Proc Baby()

Pute()
Printe("Hallo nieuwe Aardbewoner!")

Return

;***

Proc Methus()

Pute()
printe("Hallo Methusalem!")

Return

;***

Proc Controle()

If leeftijd <1 then Baby() fi

If leeftijd >200 then Methus() fi

Return

;***

Proc Calculeer()

Resultaat=2000-1988+leeftijd

Return

;***

Proc Toon()

Pute()
Print("In 2000 ben U ")
Printc(resultaat)
Printe(" jaar oud.")

Return

;***

Proc Wacht()

Pute()

Printe("Druk een toets voor einde.")
Key=255 While key=255 do od

Return

;***

Proc Main()

     Setup()
     Vraag()
     Controle()
     Calculeer()
     Toon()
     Wacht()

Return
```

'We schrijven geen Houtje-touwtje programma's'

taal leent zich daar niet voor en onszelf doen we er geen plezier mee. Eerst alles op papier zetten heeft als nadeel dat tegen de tijd dat we er mee klaar zijn, ons de lust tot programmeren alweer vergaan is. Alhoewel het voor hele grote programma's soms wel aan te bevelen is.

Hak het in stukjes
Een andere methode die ik zelf gebruik en waar ik veel plezier van heb, is de volgende. Zet in grote lijnen op papier wat het programma moet (hoort te) doen. Hak daarna het programma in stukjes (modules) en begin met het programmeren van de verschillende modules. Ik stel voor om eens te bedenken hoe een programma eruit zou moeten zien dat onze leeftijd in het jaar 2000 uitrekent. Wat moet er allemaal in zo'n programma zitten? Uiteraard een titelscherm dat ons vertelt hoe het programma heet. Dan moeten we onze leeftijd weten om uit te rekenen hoe oud we zullen zijn in het jaar 2000. Een routine om het antwoord op het scherm te tonen sluit ons programma af. Laten we eens stap voor stap door de listing lopen en kijken hoe ik dit probleem heb aangepakt. De listing begint met wat informatie over het programma. Hoe het heet, wie schreef het en wanneer was dat. Altijd doen, dat maakt het later een stuk makkelijker als je iets wilt opzoeken en meerdere listings door moet kijken.

De variabelen
Hierna zijn de viariabelen aan de beurt. Declareer in eerste instantie alleen die variabelen waarvan je zeker weet dat je ze nodig hebt in het programma. Kies "verstaanbare" namen voor de variabelen. Zo klinkt leeftijd heel wat begrijpelijker dan iets als xyzchat of abacadabra. Alfabetiseer de variabelen. Staan er veel variabelen in een programma, dan vind je ze sneller terug. Nu het programma zelf. We beginnen bij de procedure die als eerste wordt gestart, namelijk de procedure main(). Main() is het hart van ons programma waarvan uit de andere procedures worden aangeroepen. Als eerste is dat de procedure setup(). Setup() zet de titel van ons programma op het scherm, zorgt dat de kleuren goed staan (1) en vraagt hoe oud we zijn in 1988. Met vraag() halen we het antwoord hierop binnen.

Zinnige antwoorden
Nu volgt er een belangrijke procedure, controle(). Controle() kijkt of we een zinnig antwoord hebben gegeven. Vullen we als antwoord nul in dan worden we welkom geheten als nieuwe aardbewoner. Is ons antwoord groter dan 200, dan denkt het programma dat we Methusalem zijn. Dit komt misschien niet serieus over maar toch is het zo bedoeld. Als we een gebruiker van het programma een vraag stellen, dan moeten we er niet blind van uitgaan dat hij of zij deze ook goed beantwoordt. Hij of zij kan al dan niet moedwillig iets heel anders intikken dan het programma verwacht. Met als mogelijk gevolg dat het programma vastloopt. Altijd een antwoord controleren voordat je er iets mee doet! Met het antwoord gaan we vervolgens aan het rekenen in de procedure calculeer().

Liever geen do-od lus
Is het antwoord berekend, dan zet de procedure toon() dit op het scherm. Om te voorkomen dat het programma meteen stopt, roepen we de procedure wacht() aan. Deze procedure wacht tot er een toets wordt gedrukt en beëindigt dan het programma. Zonder wacht() zouden we geen tijd hebben om de uitkomst van het programma te lezen. Het programma is nu klaar en hopelijk weten we allemaal hoe oud we in het jaar 2000 zijn.

'Als een gebruiker moet wachten, gebruik dan geen do-od loop'

Door de structuur die we aangebracht hebben in het programma zijn we in staat het op een snelle wijze aan te passen of uit te breiden. Zo kunnen we bijvoorbeeld een procedure invoegen die naar onze naam vraagt, zodat het programma een persoonlijker tintje krijgt. Hiervoor hoeven alleen de procedures Setup(), vraag() en toon() aangepast te worden.

Wat losse tips. Als een gebruiker moet wachten op iets, bijvoorbeeld het lezen van tekst, gebruik dan geen do-od loop. Verstandiger is om de procedure wacht() te gebruiken. Als iemand het programma voor de zoveelste keer gebruikt, dan kan hij of zij de tekst wel dromen. Zou je een do-od loop gebruiken, dan wekt dat alleen maar irritatie op. Zet de procedures op een logische manier achter elkaar. In de listing staan alle procedures in de volgorde waarin ze aangeroepen worden. Zo weet je dat de eerste procedure van main (setup) bovenaan het programma staat. Dat voorkomt overbodig zoeken. Zo staan baby() en methusalem() in de "goede" volgorde boven controle(). Ik hoop dat deze tips gebaseerd op mijn ervaring als Action! programmeur het makkelijker en aangenamer maken om in Action! te programmeren. De volgende keer gaan we nader kennismaken met de INPUT routines.

Happy Bytes, Theo Okhuijsen

(1) Nog een voorbeeld van "verstandig" programmeren. De standaard kleuren zijn wit op blauw. Op een kleurenscherm is dit prima te lezen. Maar op een zwart-wit tv of een groene monitor is dat wat wazig te lezen. Door de achtergrond op zwart te zetten en de letters op wit is de tekst zowel in kleur als in monochroom goed te lezen. Ga er nooit vanuit dat een ander met exact dezelfde apparatuur werkt als jezelf.

P.S. Let wel, het is een voorstel wat ik hier doe en geen wet van Meden en Perzen. Als je op je eigen manier aangenamer programmeert, blijf dat dan doen. Belangrijk is dat je plezier hebt in het programmeren!

Programmeren in Action!, deel 12

Input in Action!

Deze keer gaan we het hebben over input in Action! Onder input verstaan we het binnenhalen van gegevens van verschillende aard. We kunnen de volgende soorten gegevens onderscheiden:
a: Numerieke gegevens, dit zijn getallen.
b: String gegevens, dit zijn "woorden of zinnen".

Voordat ik verder ga met input moet ik eerst even uitleggen wat Action! onder een string verstaat. Een string is een array van maximaal 255 elementen waarvan de lengte in het eerste element van de array staat. Een element van een string is van het type byte. Een voorbeeld ter verduidelijking. Voorbeeld: Byte array string="Hallo allemaal"

Dit is een string met een lengte van 14 elementen. Nu staat er in string(0) het getal 14 en in string(1) de letter H. Het eerste element van "Hallo ..." begint dus op element 1 van de array in plaats van op element 0. Verwarrend is ook dat het tellen niet bij nul begint zoals gebruikelijk maar bij een. Let hier goed op, want als je met eigen routines een string gaat bewerken en meent dat in string(0) het eerste element van de string staat, dan krijg je een nare verrassing!

Na deze uitleg over hoe een string in elkaar zit dan nu de input routines. Als we de listing bekijken, dan zien we weer de gebruikelijke manier om een Action! programma op te bouwen. Via main worden de procedures aangeroepen die we een voor een zullen uitleggen. In setup wordt het scherm geinitialiseerd, het keyboard register schoon geveegd en de titel van het programma op het scherm geprint. Daarna wordt de procedure numeriek aangeroepen waarin een voorbeeld wordt gegeven van numerieke input. In deze procedure krijgt de byte variabele getal een waarde door de functie inputb(). Inputb() wacht tot er een getal is ingetoetst afgesloten door een return. Maar wat als je een card of integer wilt opvragen? Geen paniek, voor elk type variable bestaat een bijhorende input functie. Voor een byte is dat inputb(), voor een card is dat inputc() en

```
;Input routines in Action!
;Action listing deel 12
;(c) 1988 Theo Okhuijsen
;7 november 1988, 15:31 pm
;Exclusief voor Atari Magazine

Byte getal,key=764,letter
Byte array buffer(128)

;***

Proc Setup()

Graphics(0)
Setcolor(1,0,14)
Setcolor(2,0,0)
Key=255
Poke(752,1) ;geen cursor
Position(7,2)
Printe("Action! Input Routines Demo")

Return

;***

Proc Numeriek()

Pute() ;Lege regel
Print("Voer een getal in >")
Getal=Inputb()
Print("Het getal is ")
Printbe(getal)
Pute()

Return

;***

Proc String()

Pute()
Print("Wat is de beste computer?")
Inputs(buffer)
Print(buffer)
Printe(" is de beste computer!")
Pute()
Return

;***

Proc Karakter()
```


```
 Pute()
 Print("Type een letter >")
 Close(1)
 Open(1,"K:",4,0)
 Letter=Getd(1)
 Close(1)
 Pute()
 Print("U typte de letter ")
 Put(letter)

 eturn

;***

Proc Wacht()

Position(7,22)
Print("Druk s.v.p. op een toets.")
While key<>255 do od ;tegen "luie" vingers
While key=255 do od

Return

;***

Proc Main()

      Setup()
      Numeriek()
      String()
      Karakter()
      Wacht()
      Graphics(0)
Return
```

voor een integer is dat inputi(). Hierna is de procedure string aan de beurt. In de procedure string wordt de byte array buffer gevuld met behulp van de procedure inputs().

Tot nu toe is het allemaal niet echt moeilijk en denk ik dat zowel numeriek als string geen problemen zullen geven. Zeker niet als je in string Atari als antwoord intypt. Dan is de beurt aan een lastige procedure, karakter. In karakter maken we kennis met het begrip channels wat kanalen betekent. Ik zal hier niet dieper op ingaan omdat het begrip channel samen met het OPEN statement de volgende keer uitgebreid aan de beurt komt. Karakter opent een channel naar het toetsenboard en wacht met behulp van de functie getd() tot er op een toets is gedrukt. De procedure wacht tenslotte zorgt ervoor dat we eerst op een toets moeten drukken alvorens het programma beëindigd wordt. Dit geeft ons de gelegenheid om op ons gemak het resultaat van het programma te bewonderen. Ik heb deze keer met opzet niet overal de puntjes op de i gezet. Het onderwerp is tenslotte niet echt moeilijk en wordt in de Action! handleiding ook op een behoorlijke manier uitgelegd. Tot slot van het verhaal over input moet ik vermelden dat er nog meer input procedures en functies zijn, die allemaal te maken hebben met input via een channel. Deze komen de volgende keer in het verhaal over channels en het OPEN statement aan bod.

Lezersvragen

Dan waren er nog wat vragen van lezers. Zo vroeg Antoon Linderman zich af of hij iets zou hebben aan de Action! disk zonder dat hij over de Action! cartridge beschikt. Het antwoord daarop is helaas nee. Op de Action! disk staan onder andere source files die je met behulp van de cartridge compileert voordat ze gerund kunnen worden. Zonder de cartridge heb je dus niet veel aan de PD disk. Verder had iemand, wiens brief ik helaas kwijt ben (foei!), eens in een object file lopen spieken en daar volgens hem wat vreemds ontdekt. Excuses, dat nu het even technisch wordt. Hij schreef, als je een procedure aanroept, dan wordt dat door de compiler vertaald in een JSR adres procedure. Als je op dat adres gaat kijken, vind je een JMP naar precies een locatie verder, waar dient deze constructie voor? Tsja, daar moest ik even voor op mijn kruin krabben. Na enig speuren vond ik het antwoord en het is eigenlijk heel simpel. Declareer maar eens wat lokale variabelen in een procedure. Waar denk je dat deze variabelen in het geheugen staan? Precies, tussen de JMP en de rest van de procedure. De JMP wordt dus gebruikt om als het ware over deze lokale variabelen heen te springen.

Het zit er weer op, de volgende keer gaan we het OPEN statement bespreken.
Happy Bytes,

Theo Okhuijsen

Vragen? schrijf naar: SAG Action! Postbus 2095, 5202 Cb Den Bosch.

Programmeren in Action! deel 13

Het OPEN-statement

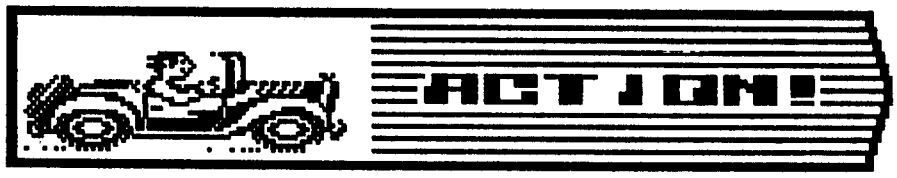
Voor de communicatie met randapparatuur beschikt de Atari XL/XE over een aantal systeemroutines. Onder randapparatuur verstaan wij (en de XL/XE) apparaten als printer, diskdrive en cassetterecorder. Maar ook het scherm is volgens het Operating System een device.

Theo Okhuijsen

Toen Atari de 400/800 (voorganger XL/XE) ontwikkelde, koos het voor een aanpak die sterk verschilde van de rest van de fabrikanten. In die dagen (1979) was het normaal dat als men een tekst wilde printen, eerst een half uurtje gepoked moest worden voordat er via een systemcall geprint kon worden. Niet bepaald gebruikersvriendelijk en begrijpelijk dat Atari voor een andere aanpak koos.
De oplossing die uit de bus kwam, was de Central Input/Output-routine die samen met de Serial Input/Output-routine alle communicatie van en naar de randapparatuur verzorgt. Met communicatie wordt bedoeld het transport van data die uit een of meerdere bytes kunnen bestaan. Het begrip randapparatuur vatte Atari zeer ruim op. Niet alleen fysieke apparaten zoals een diskdrive of cassetterecorder maar ook logische apparaten zoals het grafische scherm of de editor beschouwden de Atari-engineers als een device. Vergelijk hiermee de Commodore 64 waar je moet poken om op het scherm te kunnen tekenen.
Om de CIO/SIO te vertellen wat het moet doen heeft ieder device een eigen handler die deze taak verricht. Alle handlers hebben een aantal eigenschappen gemeen:

Communicatie met het device verloopt via CIO;
Commando's aan CIO worden via een parameterblok (IOCB) gegeven;
Een standaard set commando's voor alle devices;
Een standaard set fout-codes voor alle devices;
Een device kan (en mag) over extra commando's beschikken.

Door deze opzet worden alle devices op dezelfde manier aangesproken, zodat maar een set routines geleerd hoeft te worden. Omdat dit de Action!-rubriek is ga ik niet nader in op CIO/SIO en IOCB's, maar verwijs voor nadere informatie hierover naar de literatuurlijst.



Hoe communiceer je in Action! met een device? Daarvoor moet je een aantal dingen weten:

1: Met welk device willen je praten?
2: Wat willen je doen? Lezen of schrijven van data?
3: Waar staan de data of moeten de data komen te staan?
4: Hoe veel data zijn er? (Alleen nodig bij het XIO commando).

De tabel onderaan dit artikel geeft een overzicht van de gangbare devices met hun naam.

De syntax (opbouw) van het Action! open-statement is als volgt:

Open(channel,"Device specifier",mode-,optional byte)

Met channel wordt het nummer bedoeld van het IOCB dat we willen gebruiken. Er zijn acht IOCB's, zodat we acht verschillende devices tegelijk kunnen gebruiken. Enkele IOCB's zijn gereserveerd voor systeem-gebruik, namelijk:

IOCB 0 wordt door het E: device gebruikt.
IOCB 6 wordt door het S: device gebruikt.
IOCB 7 wordt door Action! gebruikt.

De Device specifier is de naam van een device uit tabel 1.

Mode kan de volgende waarden hebben:
4: Lees data
6: Lees directory (alleen diskdrive)
8: Schrijf data
12: Lees en schrijf data
Sommige devices gebruiken andere waarden, zie hiervoor ook de boeken en artikelen waarnaar verwezen wordt in de literatuurlijst.

Het optional byte wordt door sommige devices gebruikt om een speciale taak te verrichten. Het S: device gebruikt het om de graphics mode mee aan te geven. In de listing is dit byte 0 en daarom wordt er een graphics 0 scherm gemaakt. Geef die byte eens een andere waarden (0-15) en kijk wat er gebeurt.
In de listing wordt in de procedure Open_Dev het S: device geopend in de mode schrijven. Hebben we een device geopend, dan is het gekozen channel (IOCB) gereserveerd voor gebruik door dat device. Een ander device mag geen gebruik maken van dat IOCB!
In de procedure Write_Dev wordt twee maal de array tekst naar het scherm geschreven. De lengte van deze array staat in het eerste element en daar maken we handig gebruik van. (Pas op, het tellen begint bij nul!) De array wordt karakter voor karakter naar het scherm geschreven met behulp van de PutD() routine uit de Action!-library. PutD() stuurt een (1) byte naar het gekozen channel (IOCB). Dat is ook de betekenis van de eerste parameter van PutD(). Action! geeft zelf aan CIO door waar de data staan en hoe lang deze zijn en bespaart ons hiermee een hoop werk. Zouden we het XIO-commando gebruiken, dan zouden we veel meer parameters op moeten geven en zelf moeten uitrekenen hoe lang de array is.
Ik heb als voorbeeld voor de PutD() routine gekozen, maar de array had ook met de PrintD() routine geprint mogen worden. Nadeel van PrintD() is dat een string maximaal 255 karakters lang mag zijn.
Mochten er vragen zijn, schrijf dan naar:

Stichting Atari Gebruikers,
t.a.v. Action! rubriek,
Postbus 2095,
5202 CB Den Bosch.

Happy Bytes,
Theo Okhuijsen

Literatuurlijst

Zelf ontwerpen van device drivers, M. van der Zwaan, AM 4-3 blz. 65. Zap!: de roep van CIO, Engel Nobbe, AM 4-3 blz. 80. Mapping the Atari revised edition, Compute! De Re Atari, Atari. Home Computer Operating System User Manual, Atari.

Overzicht van device namen

C: Cassette recorder
D: Disk drive
E: Editor
K: Keyboard
P: Printer
R: Rs232 (850 interface)
S: Screen

Opmerking bij de device-namen:
Het D: device heeft achter de device naam een file naam die bestaat uit acht karakters met eventueel een extender van drie karakters gescheiden door een punt. Zie Mapping the Atari voor uitgebreidere informatie over devices.

```
;Het Action! OPEN statement.
;Action! listing deel 13
;(c) 1989 Theo Okhuijsen
;15 februari 1989
;Exclusief voor Atari Magazine

Byte array tekst=
"---Action! OPEN statement demo---"

Byte flag,teller

;###

Proc Open_Dev()

Clo=#(1)
Op.  ,"S:",8,0)

Return

;###

Proc Set_Up()

SetColor(1,0,14)
SetColor(2,0,0)

Return
```

```
;###

Proc Write_Dev()

Teller=1
Flag=tekst(0)

While teller#flag+1
 Do
  PutD(1,tekst(teller))
  Teller==+1
 Od

PutDe(1)

Teller=tekst(0)
While teller#0
 Do
  PutD(1,tekst(teller))
  Teller==-1
 Od
PutDe(1) PutDe(1)
PrintDe(1,"     Druk op een toets.")

Return

;###

Proc Wait_Key()
```

```
Byte key=764

Key=255 While key=255 do od

Return

;###

Proc Clean_Up()

Close(1)
Graphics(0)

Return

;###

Proc Main()

Open_Dev()      ;Open S: device
Set_Up()        ;Zet kleuren
Write_Dev()     ;Schrijf tekst
Wait_Key()      ;Wacht op toets
Clean_Up()      ;Close device

Return
```