Seminararbeit

INTERCAL

Natalie Mair
csam8725@uibk.ac.at

16 Februar 2013

Betreuer: Bertram Felgenhauer

Zusammenfassung (Englisch)

INTERCAL is one of the oldest esoteric programming languages. It was created with the intentsion to have a compiler language which has nothing at all in common with any other major language. INTERCAL is highly esteemed under the esoteric programming languages, not only for its uniqueness and importance. To get an insight into INTERCAL is an enrichment for every programmer.

Inhaltsverzeichnis

1 Einsatzbereich und Unterschiede

1.1 Historische Entwicklung und Visionen

INTERCAL wurde am 26. Mai 1972 von Donald R. Woods and James M. Lyon, zwei Studenten der Princeton Universität, entwickelt. Etwa 18 Jahre später, also 1990, entwickelte Eric S. Raymond eine erste UNIX-Implementierung. Diese Version von INTERCAL ist heute als C-INTERCAL bekannt. Eine weitere Version CLC-INTERCAL wird von Claudio Calvelli gewartet. INTERCAL-72 wird die traditionelle Version genannt, sie ist unter anderem Turing-vollständig.

Gemäß des originalen INTERCAL-Handbuches lautet der vollständige Name des Compilers „Compiler Language With No Pronounceable Acronym“ welcher auf INTERCAL gekürzt wurde.

INTERCAL gehört zu den esoterische Programmiersprachen, und war eine, wenn nicht „die“ erste Programmiersprache, ihrer Art. Esoterische Programmiersprachen sind meist für den praktischen Einsatz ungeeignet. In einigen Fällen stecken hinter ihren Entwicklungen schlicht akademische Scherze, oder einfach nur Konzepte, die es erlauben neue Ideen umzusetzen. INTERCAL ist eine der hoch angesehenen esoterische Programmiersprache.

Um den Hintergrund der Entwicklung dieser Sprache besser zu erläutern, greife ich auf eine Auszug aus dem originalen INTERCAL-Handbuch [Manual] zurück.

The explicit design goal of INTERCAL is

„...to have a compiler language which has nothing at all in common with any other major language. By ‘major’ we meant anything with which the author’s were at all familiar, e.g., FORTRAN, BASIC, COBOL, ALGOL, SNOBOL, SPITBOL, FOCAL, SOLVE, TEACH, APL, LISP and PL/I.[Manual] “

1.2 Einsatzgebiet

Hinter der Entwicklung von INTERCAL steckt die Idee, eine völlig neue Programmiersprache zu entwickeln, die von den meisten bisher bekannten Konzepten abweicht. INTERCAL sollte komplett verschieden sein von den gängigen Programmiersprachen wie z.B. FORTRAN, BASIC, COBOL, ALGOL, SNOBOL, SPITBOL, FOCAL, SOLVE, TEACH, APL, LISP and PL/I. Ziel war es, das Programmieren so schwierig wie möglich zu gestalten. Entstehende Programme sollten außerdem unleserlich wirken. INTERCAL hat daher nur sehr wenig bis keine Ähnlichkeiten mit bekannten Programmiersprachen und ist außerdem schwer zu erlernen.

1.3 Vergleich zu anderen Programmiersprachen

INTERCAL besitzt nur wenige Gemeinsamkeiten zu anderen bekannten Programmiersprachen, sie wurde bereits mit den Gedanken entwickelt sich völlig von allen anderen Programmiersprachen zu unterscheiden.

INTERCAL entnimmt nur Konzepte wie Variablen, Arrays, Text input/output und Anweisungen aus anderen Programmiersprachen. Alle anderen Operatoren und Ausdrücke sind einzigartig und verwirrend zugleich.

INTERCAL besitzt keine einfachen if-Anweisungen für Verzweigungen, es existieren keine Schleifen und auch keine mathematischen Grundoperatoren (z.B. Addition).

INTERCAL hat einige Konzepte aus anderen Programmiersprachen übernommen und abgeändert. Der English-like Style in INTERCAL beruht auf COBOL. Alle Statements können mit PLEASE eingeleitet werden. INTERCAL Statements in diesem COBOL Style können unter anderem FORGET, REMEMBER, ABSTAIN und REINSTATE sein.

Aus FORTRAN hat sich INTERCAL einerseits die optionale Zeilennummerierung, andererseits den DO Konstrukt entnommen, welcher in FORTRAN als Schleifen-Initialisierer benutzt wird. In INTERCAL muss hingegen jedes Statement mit DO beginnen.

Wie es in APL üblich ist, bemächtigt sich INTERCAL der Nutzung von einzelnen Zeichen als bestimmte Operatoren.

In gewisser Hinsicht kombiniert INTERCAL die schlimmsten Features einiger Programmiersprachen und schafft eine neue Sprache.

2 Besondere Features

2.1 Non-Standard Features

Syntax und Semantik

Line Labels Der erste Teil eines INTERCAL-Statements ist das sogenannte **Line Label** welches die Zeilennummer des Statements spezifiziert. Das Line Label ist optional; es sind auch Statements ohne Angabe des Line Labels erlaubt, dieses unterbindet jedoch, dass andere Befehle auf dieses Statement referenzieren können. Line Labels müssen Konstanten und eindeutig in einem INTERCAL Programm sein. Allerdings müssen sie nicht in geordneter Reihenfolge angegeben werden. Ein Line Label wird durch einen Integer innerhalb von runden Klammern ausgedrückt.

Beispiel 2.1. Line Label

```
(10) DO...
(2) DO...
(16) DO...
```

Es sind nur Zeilennummern im Bereich zwischen 1 und 65.535 erlaubt, wobei jene zwischen 1000 und 1999 von der Systembibliothek benutzt werden. Diese Zeilennummern zu benutzen kann also unerwartete Fehler produzieren, wenn die Systembibliothek eingebunden ist.

Viele INTERCAL Statements benutzen Ausdrücke als Argumente. Ausdrücke bestehen aus Operanden und Operatoren zwischen ihnen. In INTERCAL gibt es keine Operatorenpräzedenz; verschiedene Compiler lösen Mehrdeutigkeiten auf verschiedene Arten. Einige Compilerversionen, inklusive des originalen INTERCAL-72 Compilers, generieren Fehlermeldungen während des Kompilierens oder Ausführens mehrdeutiger Ausdrücke. Daher ist es das Sicherste, alle Ausdrücke durch Klammerung zu gruppieren.

Statementindikator Einem Line Label, folgt ein Statementindikator, welcher den Start eines Statements markiert.

Der wichtigste Statementindikator ist DO, es gibt aber auch höflichere Synonyme dieses Bezeichners; PLEASE und PLEASE DO. Alle drei haben aber dieselbe Bedeutung, sie markieren den Anfang eines Statements.

Kommentare Durch konkatenieren des DO-Bezeichner, oder seiner Synonyme mit NOT oder N'T können in INTERCAL Kommentare hinzugefügt werden. Codezeilen welche in INTERCAL hinter einer Markierung DO NOT oder

DON'T stehen, werden nicht ausgeführt. Es ist jedoch möglich diesen Code mittels REINSTATE (2.1 Seite 10) trotzdem ausführbar zu machen.

Beispiel 2.2. Kommentare
```
DO NOT das ist ein Kommentar!
PLEASE DON'T Ein weiterer Kommentar!
DO NOTE this is a Comment!
```

Klammerung Da es in INTERCAL keine Präzedenz für Operatoren gibt, existieren diverse Möglichkeiten um die Präzedenz zu spezifiziern.

Alle Versionen von INTERCAL akzeptieren die INTERCAL-72 Gruppierungsregeln. Diese besagen, dass es möglich ist eine Präzedenz durch Gruppierung innerhalb von **Hochkommas (' ', sparks)** und **Anführungszeichen (" ", rabbit-ears)** festzulegen. In anderen Programmiersprachen funktioniert das Festlegen einer Präzedenz meist durch Klammerung. INTERCAL-72 und C-INTERCAL Versionen verlangen, dass Ausdrücke vollständig durch diese Art gruppiert werden. Diese Methode wird auch immer noch empfohlen, da sie zu portableren Programmen führt und leichter zu verstehen ist. Ob Sparks oder Rabbit-ears verwendet werden ist egal, der Programmierer kann der Übersichtlichkeit oder der Ästhetik halber die einen oder die anderen benutzen. Eine Technik ist es Sparks auf dem äußersten Level der Gruppierung zu verwenden, dann Rabbit-ears in nächsten Level und wieder Sparks im übernächsten usw.

Beispiel 2.3. Klammerung
V(&(a$b)) wir in INTERCAL so geklammert: ‘V “&a$b“ ‘ oder “V ‘&a$b‘ “

Konstanten Konstanten können nur als 16-bit Werte und nur im Bereich zwischen 0 und 65.535 dargestellt werden. Sie werden durch ein Doppelkreuz gekennzeichnet (# „mesh“).

Beispiel 2.4. Konstanten
#256

Variablen INTERCAL erlaubt nur die Verwendung von 2 verschiedenen Typen von Variablen, 16-bit integer und 32-bit integer. Diese werden durch einen Punkt (. „spot“) und einen Doppelpunkt (: „two-spot“) gefolgt von einer Zahl zwischen 1 und 65.535 dargestellt. Diese Variablen enthalten nur nicht-negative Werte, sodass der Wertebereich jeweils zwischen 0 und 65.535 bzw. 0 und 4.294.967.295 liegt. Vorsicht ist dennoch geboten, da z.B. .123 und :123 zwei verschiedene Variablen darstellen, .1 und .0001 jedoch identisch sind. Variablen müssen nicht deklariert werden.

Beispiel 2.5. Variablen
.123

Arrays Arrays können durch einen Beistrich (, „tail“), für 16-bit Werte, oder durch einen Strichpunkt (; „hybrid“), für 32-bit Werte, gefolgt von einer Zahl zwischen 1 und 65.535 dargestellt werden. Zunächst muss die Dimension des Arrays festgelegt werden. Anschließend kann mittels SUB-Anweisung, auf die einzelnen Array-elemente zugegriffen werden. Auch hier gilt wiederum ,123 ist verschieden zu ;123.

Beispiel 2.6. Eindimensionale Arrays
DO ,1 <- #13
DO ,1 SUB #1 <- #238
DO ,1 SUB #2 <- #108
DO ,1 SUB #3 <- #112

Arrays können durchaus auch mehrdimensional sein, dazu werden die einzelnen Dimensionen mittels des BY-Operators getrennt.

Beispiel 2.7. Mehrdimensionale Arrays
DO ,1 <- #2 BY #3
DO ,1 SUB #1 #2 <- #12

Operatoren INTERCAL kennt 5 Operatoren, 2 binäre und 3 unäre und benutzt nur diese um Berechnungen durchzuführen.

Binäre Operatoren Die binären Operatoren werden als **Interleave** ($ „big money“) und **Select** (∼ „sqiggle“) Operatoren bezeichnet.

Beispiel 2.8. Binäre Operatoren
DO :1 <- #0$#256

Der **Interleave-Operator** produziert sein Ergebnis durch alternieren der jeweiligen Bits der Operanden. Das heißt, er nimmt das erste Bit des ersten Operanden und stellt es an die erste Stelle des Ergebnisses, an die zweite Stelle kommt das erste Bit des zweiten Operanden. Die dritte Stelle bildet das zweite Bit des ersten Operanden und die vierte enthält das zweite Bit des zweiten Operanden, usw. Der Interleave-Operator erhält als Input zwei 16-bit Werte und produzieren ein 32-bit Ergebnis.

Definition 2.9. Interleave ($)

a = erste Operand mit $(a_{15} \dots a_0)_2$

b = zweite Operand mit $(b_{15} \dots b_0)_2$

r = Ergebnis $(a_{15} b_{15} \dots a_0 b_0)_2$

Beispiel 2.10. Interleave (\$)

$$a = (1111111111111111)_2 \tag{1}$$
$$b = (0000000000000000)_2 \tag{2}$$
$$r = (10101010101010101010101010101010)_2 \tag{3}$$

Der **Select-Operator** entnimmt dem ersten Operanden alle Bits, welche im zweiten mit 1 übereinstimmen.

Definition 2.11. Select ($\sim$)

a = erste Operand mit $(a_{15} \dots a_0)_2$

b = zweite Operand mit $(b_{15} \dots b_0)_2$

r = Ergebnis $(a_{15}$ if $b_{15} = 1 \dots a_0$ if $b_0 = 1)_2$

Beispiel 2.12. Select ($\sim$)

$$a = (a_{15} \dots a_0)_2 \tag{4}$$
$$b = (1110101010101010)_2 \tag{5}$$
$$r = (a_{15}a_{14}a_{13}a_{11}a_9a_7a_5a_3a_1)_2 \tag{6}$$

Unäre Operatoren Zu den unären Operatoren gehören das **logische UND** (& „AND“), das **logische ODER** (V „OR“) und das **logische XOR** (? „XOR“). Diese Operatoren können zwischen einem spot, two-spot, oder einem Doppelkreuz und einem Integer stehen. Das multiple Konkatenieren von unären Operatoren ist nicht erlaubt.

.&123, #?123

ungültig: ‘&&#5\$#7‘

Unäre Operatoren erhalten nur einen Inputwert, sie rotieren diesen Operanden zuerst um eine Position nach rechts, und erhalten so einen zweiten Operanden. Nun wird die logische Operation auf die beiden Operanden ausgeführt.

Definition 2.13. Unäre Operatoren

$$a = Operand(a_{15} \dots a_0)_2$$

$$a' = \mathsf{rotate\_right}(a) = (a_0, a_{15} \dots a_1)_2$$

$$r = \text{Ergebnis } f(a, b) \text{ wobei } f \in \{\&, V, ?\}$$

Beispiel 2.14. logische UND (&)

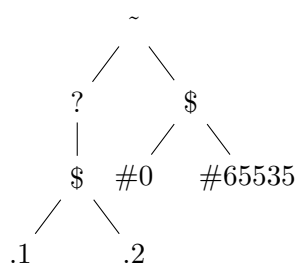
$$a = (0111111100000000)_2 \tag{7}$$
$$b = (0011111110000000)_2 \tag{8}$$
$$r = \&(a, b) \tag{9}$$
$$r = (0011111100000000)_2 \tag{10}$$

Gerade die Syntax der unären Operatoren trägt wesentlich dazu bei, dass INTERCAL-Programme relativ unleserlich wirken. Das kommt daher, dass die Schreibweise und Syntax aus der Mathematik und anderen Programmiersprachen weit verbreitet und daher viel intuitiver ist. In der Mathematik wird eine beliebige Funktion f über zwei Variablen a un b wie folgt notiert: f(a,b). In INTERCAL wird das Funktionssymbol in die Klammerung hineingezogen und würde demnach durch (fa,b) dargestellt werden. INTERCAL verwendet an Stelle von Klammern Hochkommas, daraus ergibt sich der Ausdruck 'fa,b'. Eine binäre UND-Operation auf zwei Variablen definiert man in der Mathematik also durch &(a,b), die Schreibweise in INTERCAL hingegen sähe folgendermaßen aus: '&a,b'.

Statements INTERCAL stellt eine sehr große Palette an **Statements** zur Verfügung. Einige gehören dabei zu INTERCAL-Dialekten andere wiederum sind Teil der Kernsprache. Zahlreiche Statements haben leicht unterschiedliche Effekte in den verschiedenen Implementationen von INTERCAL. Darum ist es immer wichtig, ein Programm auf mehreren Compilern zu testen, um sicherzustellen, dass es portabel ist.

Berechnung Der einzige INTERCAL Befehl, der kein Keyword beinhaltet, ist bekannt als „Calculate"-Befehl. Dabei werden Variablen, Arrayelementen oder Arrays Werte zugewiesen.

Beispiel 2.15. .3 <- '?.1$.2'~'#0$#65535'



$$.1 = (a_{15} \dots a_0)_2$$
$$.2 = (b_{15} \dots b_0)_2$$
$$.1\$.2 : (a_{15}b_{15} \dots a_0b_0)_2$$
$$rotate : (b_0a_{15}b_{15} \dots b_1a_0)_2$$
$$? : (c_{31} \dots c_0)_2$$
$$\#0 = (0_{15}...0_0)_2$$
$$\#65535 = (1_{15}...1_0)_2$$
$$\#0\$\#65535 : (0_{15}1_{15}...0_01_0)_2$$
$$\sim: (c_{30}c_{28} \dots c_2c_0)_2$$
wobei $c_{2k} = b_k \ ? \ a_k$;

berechnet wird also das das bitweise XOR aus .1 und .2

NEXT Die einzigen Kontrollflussbefehle in INTERCAL-72 sind NEXT, FORGET und RESUME. Zusammen manipulieren sie den sogenannten NEXT-Stack, der auf 80 Stellen limitiert ist. Das NEXT Statement nimmt ein Line Label als Argument, und sein ausführen bewirkt den Transfer des Programmflusses zur Referenz des Line Labels. Dazu speichert diese Statement, die Position unmittelbar nach dem NEXT Befehl im Programm, an die oberste Stelle des NEXT-Stacks.

Listing 1: Addition aus der Standardbibliothek. (Auszug)

```
(1004)  PLEASE FORGET #1
        ...
        PLEASE DO .5 <- "?!6~#32768'$#1"~#3
        DO (1002) NEXT
        DO .4 <- #2
        ...
(1002)  DO (1001) NEXT
                ...
(1001)  DO RESUME .5
```

FORGET und RESUME FORGET und RESUME nehmen Ausdrücke als Argumente. Um zu verhindern, dass sich der NEXT-Stack überfüllt, ist es möglich Elemente durch FORGET und RESUME vom Stack zu entfernen. Beide entfernen eine Anzahl an Elementen gleich der Nummer ihrer Argumente. RESUME transferiert zudem den Programmfluss zu der letzten vom Stack entfernten Position.

Traditionell werden Booleanwerte in INTERCAL Programmen als #1 und #2 gespeichert. Das kommt daher, dass der einfachste Weg eine if-Anweisung

in INTERCAL-72 zu implementieren ist durch NEXT, erneut NEXT, dann RESUM entweder auf 1 oder 2 abhängig vom Argument, und schließlich wenn das Argument durch 1 evaluiert wurde, wird FORGET auf die restlichen NEXT-Stackelemente ausgeführt.

STASH und RETRIEVE Jede in einem INTERCAL Programm benutzte Variable besitzt ihren eigenen Stack, welcher Werte vom gleichen Typ beinhalten kann. STASH erlaubt das Ablegen eines Variablenwertes auf dessen Stacks. RETRIEVE liefert den obersten Wert des Stacks zurück und ersetzt ihn an Stelle des aktuellen Variablenwertes.

Beispiel 2.16. STASH und RETRIEVE
```
DO STASH .1 + ;2
DO RETRIEVE ,3
```

Es ist außerdem auch möglich mehrere Variablen gleichzeitig abzulegen oder zurückzuholen. Dazu werden deren Namen mittels Intersection(+) verbunden. Außerdem ist es möglich, dieselbe Variable mehrmals im einem Statement abzulegen oder zurückzuholen.

GIVE UP Das GIVE UP Statement führt zur Beendigung des Programmes, oder in einem Multithreaded (3.2 Seite 13) Programm führt es zur Beendigung des aktuellen Threads. GIVE UP ist die einzige Möglichkeit ein Programm ohne Fehler zu beenden.

IGNORE und REMEMBER In INTERCAL sind entweder Lese- und Schreib- oder nur Lese-Zugriffe auf Variablen möglich. Beim Start eines Programmes haben alle Variablen Lese- und Schreibrechte, diese können aber während der Programmausführung dynamisch verändert werden. Ermöglicht wird dies durch die Statements IGNORE und REMEMBER, die Syntax ist dieselbe wie zuvor bei STASH und RETRIEVE und auch hier können Variablen mittels Intersection verbunden werden.

Beispiel 2.17. IGNORE und REMEMBER
```
DO IGNORE .4
DO REMEMBER ,4 + ;5
```

Das IGNORE Statement beschränkt eine Variable auf nur lese Zugriffsrechte, REMEMBER hebt diese Beschränkung auf und ermöglicht wieder vollen Zugriff auf die Variable. Jeder Versuch einer Wertzuweisung einer nur-lese Variable schlägt fehl. Benutzt werden diese Statements vielfach in der Systembibliothek um die Veränderung bestimmter Variablen zu verhindern.

ABSTAIN und REINSTATE Der Statementindikator entscheidet ob ein Statement beim Programmstart im Zustand „Abstain“ oder „Reinstate“ ist. Diese Zustände entscheiden, ob das Statement komplett durchlaufen wird. Es ist jedoch möglich, den Zustand dynamisch während der Programmausführung zu verändern.

2.2 Beispiel

Listing 2: Hello World program in INTERCAL.

```
/*INTERCAL helloworld.i*/
DO ,1 <- #13
PLEASE DO ,1 SUB #1 <- #238
DO ,1 SUB #2 <- #108
DO ,1 SUB #3 <- #112
DO ,1 SUB #4 <- #0
DO ,1 SUB #5 <- #64
DO ,1 SUB #6 <- #194
DO ,1 SUB #7 <- #48
PLEASE DO ,1 SUB #8 <- #22
DO ,1 SUB #9 <- #248
DO ,1 SUB #10 <- #168
DO ,1 SUB #11 <- #24
DO ,1 SUB #12 <- #16
DO ,1 SUB #13 <- #162
PLEASE READ OUT ,1
PLEASE GIVE UP
```

Beispiel für J-INTERCAL 0.11, J-INTERCAL 0.12, C-INTERCAL 28.0

INTERCAL ist eine der Sprachen, in der sogar das schreiben des „Hello, World!“ Programms fast schon eine Folter ist. Das kommt daher, dass die READ OUT-Anweisung auf Turingmaschinenbasis funktioniert. Um diese Anweisung nutzen zu können, muss das Outputargument in ein eindimensionales Array (,1) gespeichert werden. Die Werte des Arrays werden eine nach dem anderen, von links nach rechts, ausgegeben.

Um das Programm erfolgreich zu kompilieren, ist eine bestimmte „Höflichkeit“ (3.1 Seite 12) des Programms notwendig. Darum muss dieses Programm vier oder fünf PLEASE-Anweisungen enthalten. Dabei ist es egal an welchen Stellen im diese platziert werden.

Andere Anweisungen hingegen sind trivial: # ist das Präfix für Konstanten, <- ist die Anweisung, SUB ist das Subscript eines Arrays. Beispielsweise als Erläuterung für die erste Zeile, ,1 ist ein Array von 16-bit Integervariablen und

es wird 13 Elemente enthalten. Jeweils fünf für 'Hello' und für 'World' und anschließend noch einen für „,“ „!“ und das Leerzeichen.

Noch besser drückt das Programm 99_bottles_of_bee [99-bottles-of-beer] die Komplexität von INTERCAL aus. Während das Programm in Java etwa an die 90 Zeilen Code umfasst, ist es in INTERCAL um die 300 Zeile lang.

2.3 Meinung der Entwickler

Laut Meinung der Entwickler ist INTERCAL eine sehr unkomplizierte Programmiersprache, wegen ihrer strikten Einfachheit. Sie hat nur wenige Grundfähigkeiten, welche relativ problemlos zu merken sind. Nach Donald R. Woods und James M. Lyon ist es sehr unproblematisch INTERCAL zu erlernen, und sie sind auch davon überzeugt, dass sich diese Sprache besonders dazu eignet neue Programmierlehrlinge auszubilden. Die Entwickler finden unter anderem, dass sich diese Programmiersprache auch als Herausforderung für professionelle Programmierer eignet. INTERCAL bietet eine unerschöpfliche Kapazität für Programmierbegeisterte, Programmiershopmanager zu verwirren, neue Freunde zu gewinnen und Personen zu beeinflussen. Da man z.B. mit Hilfe von INTERCAL behaupten kann, die einfachste Methode einen Wert von 65.536 in eine 32-bit Variable zu speichern sei:

```
DO :1 <- #0$#256
```

3 Esoterische Programmiersprache

3.1 C-INTERCAL Compiler

Der C-INTERCAL Compiler, auch „ick“ genannt, kompiliert INTERCAL-Programme und überprüft sie auf ihre Höflichkeit. Ein INTERCAL-72 Feature welches nicht im originalen Handbuch dokumentiert ist, ist die Höflichkeit eines Programms. INTERCAL erfordert ein gewisses Level an Höflichkeit des Programmierers, welche durch den PLEASE-Qualifizierer ausgedrückt werden kann. Wenn weniger als 1/5 der Programmstatements den PLEASE-Qualifizierer enthalten, reagiert der Compiler und verlangt angemessene Höflichkeit.

Sollten nun weniger als 1/5 der Programmstatements den PLEASE-Qualifizierer enthalten, wird der Compiler über eine nicht ausreichende Höflichkeit klagen.

```
mairn@mairn-R580-R590:~$ ick helloworld.i
ICL079I PROGRAMMER IS INSUFFICIENTLY POLITE
        ON THE WAY TO 20
        CORRECT SOURCE AND RESUBNIT
mairn@mairn-R580-R590:~$
```

Abbildung 1: Compilerfehler bei weniger als 1/5 PLEASE Anweisungen

Sollten nun mehr als 1/3 der Programmstatements den PLEASE-Qualifizierer enthalten, wird der Compiler über übertriebene Höflichkeit klagen.

```
mairn@mairn-R580-R590:~$ ick helloworld.i
ICL099I PROGRAMMER IS OVERLY POLITE
        ON THE WAY TO 20
        CORRECT SOURCE AND RESUBNIT
mairn@mairn-R580-R590:~$
```

Abbildung 2: Compilerfehler bei mehr als 1/3 PLEASE Anweisungen

3.2 Spracherweiterungen

TriINTERCAL Die Autoren von C-INTERCAL entwickelten später eine komplexere Version von INTERCAL unter dem Namen TriINTERCAL. TriINTERCAL verwendet an Stelle des Binärsystem das Ternärsystem, oder auch Dreiersystem. Es ist ein Stellenwertsystem zur Basis 3 und verwendet die Ziffern 0,

1 und 2. Hinter der Entwicklung von TriINTERCAL steckt unter anderem der Gedanke, dass die Zahl 65.535 relativ „bekannt“ $(111111111111)_2$ war, aber der Wert 59.048 $(2222222222)_3$ hingegen nicht.

TriINTERCAL Programme unterscheiden sich von INTERCAL Programmen in der Dateiendung .3i zum Unterschied zu .i. TriINTERCAL führt zudem einen neuen Operatoren ein, den BUT-Operator. Der Sharkfin- und der What-Operator stellen zwei verschiedene Interpretationen des XOR-Operators dar.

INTERCAL wurde über die Jahre hinweg erweitert und es wurden neue Features hinzugefügt. Eines der bemerkenswertesten Features ist das COME FROM Statement. Wenn ein COME FROM Statement auf ein anderes Statement referenziert, wird immer, wenn dieses Statement erreicht ist, der Kontrollfluss des Programmes zum COME FROM transferiert, nachdem das Statement fertig ausgeführt wurde.

Beispiel 3.1. COME FROM
COME FROM (time-reversed GOTO)
DO COME FROM (label)

Threaded INTERCAL Threaded INTERCAL ist eine Erweiterung von INTERCAL von Malcolm Ryan die die Verwendung von Threads ermöglicht.

Mehrfache Verwendung des COME FROM Statements auf das selbe Line Label bewirken, dass das Programm separate Threads für jedes COME FROM Statement startet. ABSTAIN und REINSTATE Codezeilen eines Threads beeinflussen in Threaded INTERCAL alle anderen Threads. Eine Synchronisation wird durch die Verwendung der neuen Statments DO...ONCE und DO...AGAIN erreicht.

3.3 Random Compiler Bug

Kein Compiler ist perfekt, manchmal ist es möglich, dass es zu zufälligen Fehlern kommen kann. In diesem Fall kommt es in INTERCAL zu folgendem Fehler: E774 RANDOM COMPILER BUG. Es gibt aber eine einfache Möglichkeit, dem Auftreten dieses Fehlers vorzubeugen, durch verwenden der Option -b. Die Möglichkeit diesen Bug auszuschalten stammt aus dem originalen INTERCAL-72 Compiler und wurde darum in den C-INTERCAL Compiler übernommen. Zum anderen dient diese Option als Belohnung für jene, die das Handbuch lesen.

3.4 double-oh-seven

In INTERCAL besteht die Möglichkeit einen Befehl so zu spezifizieren, dass er nur zu einer gewissen Wahrscheinlichkeit ausgeführt wird. Dies ist ein nur sehr selten genutztes Feature von INTERCAL, dafür ist es aber die einzige Möglichkeit, Zufälligkeit in ein Programm zu bringen. Der C-INTERCAL Compiler approximiert dies mit Pseudozufälligkeit. Eine Zufallszuweisung wird unmittelbar nach dem Statementindikator, aber vor dem Rest des Statements festgelegt. Diese wird durch ein sogenanntes „double-oh-seven“ (%) gefolgt von einem Integer zwischen 1 und 99 festgelegt. Der Integerwert legt also die Wahrscheinlichkeit in Prozent fest, wie groß die Chance ist, dass ein Statement ausgeführt werden soll.

Beispiel 3.2. double-oh-seven (%)

```
DO %40 WRITE OUT #1
```

Im obigen Beispiel soll die Konstante #1 ausgegeben werden, in diesem Fall hat dieses Statement eine 40 prozentige Wahrscheinlichkeit „I“ auszugeben. Das double-oh-seven ist nicht ermächtigt ein DON’T Statement zu überschreiben, d.h. das Statement „DON’T %40 WRITE OUT #1“ hätte keine Chance ausgeführt zu werden, weil das DON’T Statement eine Ausführung des darauf folgenden Codes verhindert.

4 Zusammenfassung

4.1 Computerworld Interview mit Don Woods

Has anyone ever accidentally taken INTERCAL to be a serious programming language?
Heavens, I hope not! (Though I was concerned YOU had done so when you first contacted me!)

In your opinion, has INTERCAL contributed anything useful at all to computer development?
Does entertainment count? :-)

4.2 Wurde das Ziel erreicht?

Donald R. Woods und James M. Lyon entwickelten INTERCAL mit dem Ziel, eine Programmiersprache zu schaffen, ohne jegliche Gemeinsamkeiten zu jeder existierenden Programmiersprache. INTERCAL war darin größtenteils erfolgreich, und hat zudem auch andere Programmierer dazu inspiriert, esoterische Programmiersprachen zu entwickeln. INTERCAL war sicherlich einer der Grundsteine in der Entwicklung von esoterischen Programmiersprachen.

Literatur

[99-bottles-of-beer] http://99-bottles-of-beer.net/language-intercal-333.html

[Manual] http://www.muppetlabs.com/ breadbox/intercal-man/ Woods, D. and J. Lyon, The INTERCAL Programming Language Revised Reference Manual. 1st Ed. 1973, C-INTERCAL revisions, L. Howell and E. Raymond, 1996.

[wiki] http://esolangs.org/wiki/INTERCAL

[C-INTERCAL Manual] http://catb.org/esr/intercal/ick.htm

[cse.unsw.edu] http://www.cse.unsw.edu.au/malcolmr/intercal/threaded.html

[progopedia] http://progopedia.com/implementation/c-intercal/

[techworld] http://www.techworld.com.au/article/251892/a-z_programming_languages_intercal/

[wikipedia] http://en.wikipedia.org/wiki/INTERCAL