FORTRAN WORKSHOP

Organized by BCS Edinburgh Branch
and Edinburgh Regional Computing Centre
in collaboration with
BCS FORTRAN SPECIALIST GROUP

Tuesday April 6 and Wednesday April 7 1971

CONTENTS

BCS Edinburgh Branch

University of Edinburgh — Edinburgh Regional Computing Centre

FORTRAN WORKSHOP

in collaboration with

BCS FORTRAN SPECIALIST GROUP

TUESDAY, APRIL 6

9.30am TO 5.00pm

Assembly — William Robertson Building

Sub-meetings — David Hume Tower

WEDNESDAY, APRIL 7

9.30am TO 5.00pm

General meeting — William Robertson Building

GEORGE SQUARE, EDINBURGH

GENERAL INFORMATION

LOCATION

General assembly and discussion will be in the William Robertson lecture theatre.

Sub-groups will meet in the David Hume Tower.

All meetings to be held in the George Square area of the University.

TIMING

The workshop will be held from 9.30 am to 5.00 pm on both April 6 and April 7, 1971.

MEALS & REFRESHMENTS

The cost of lunch and refreshments is included in the fee.

ACCOMMODATION

Delegates are expected to make their own arrangements for accommodation.

If requested, the organizers may be able to arrange accommodation in the University Halls of Residence at £2.50 per day for dinner, bed and breakfast.

The B.C.S. Fortran specialist group established three working parties concerned with:

Extensions to ANSI Fortran
Standards for conversational Fortran
Standards for small machine Fortran

It has been decided that a Fortran Workshop take place in Edinburgh on April 6 & 7, 1971, open to all persons interested in these topics, and is to be held from 9.30 am to 5.00 pm in the William Robertson :cture theatre, George Square.

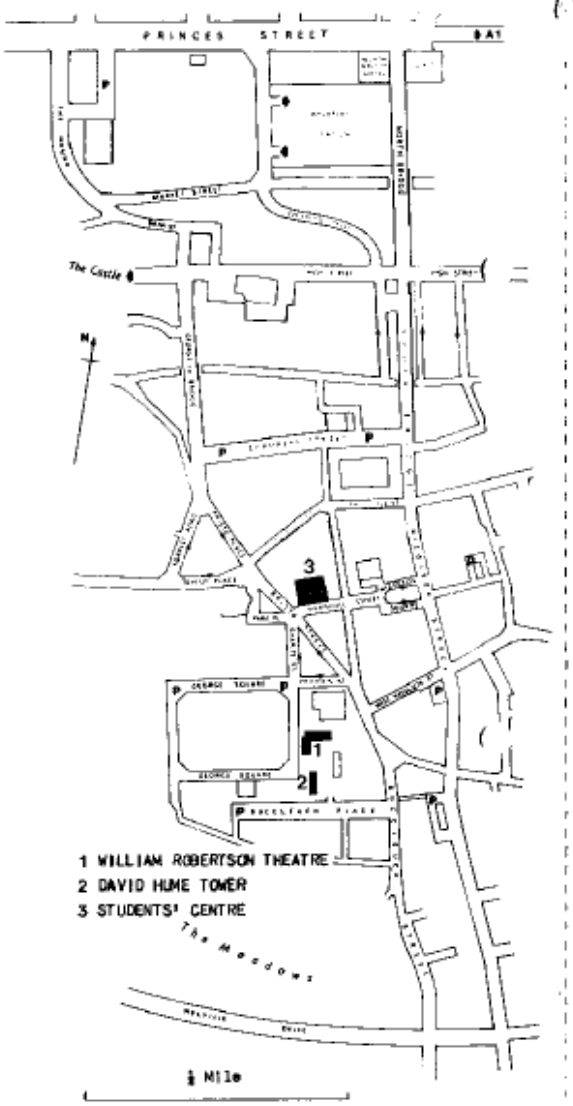
This Workshop is organized by the Edinburgh branch of the B.C.S. and ERCC, in conjunction with the B.C.S. Fortran Specialist Group.

The principal concern of the workshop will be a feasibility assessment for establishing standards for small machines and for conversational operation together with proposals for extending ANSI Fortran. It is proposed that, following this workshop, the ideas for extension be submitted to the standards committee for assessment.

The workshop will be subdivided on the first day into sub-groups concerned with the discussion of specific :ics under the chairmanship of working party members. ine reports of each sub-group will be presented to the general assembly on the following day for general discussion and debate.

While it is not intended that formal papers be presented, the workshop will be organized to allow persons holding opinions adequate time to express them. For this reason the cost of the workshop has been kept as low as possible.

Delegates are expected to make their own arrangements for accommodation.



1 WILLIAM ROBERTSON THEATRE
2 DAVID HUME TOWER
3 STUDENTS' CENTRE

CENTRAL EDINBURGH

REGISTRATION FORM

I wish to attend the FORTRAN WORKSHOP on April 6 & 7, 1971, for which I enclose the fee of £3.50 (£3-10-0)

NAME

ADDRESS/DEPARTMENT

telephone number:

To assist the organizers would you please indicate which of the following discussion group topics are of interest, with no obligation.

- Conversational ☐
- Mini-computers ☐
- Run-time diagnostics and error control ☐
- Program structure ☐
- Input/output facilities ☐
- Review of executable statements ☐
- Additional types. e.g. character ☐
- Relationship with operating system environment ☐

Alternative proposal:

This form together with the fee should be returned not later than 19 March 1971 to:

The FORTRAN WORKSHOP,
Edinburgh Regional Computing Centre,
The King's Buildings,
Edinburgh EH9 3JZ.

T/ (031) 667 1011 Ext 3613

Cheques should be made payable to the University of Edinburgh.

B.C.S. Edinburgh Branch

"FORTRAN Group" (Executable & Spec. Statements)
5th/6th April, 1971

Secretary: Mr. S. V. Dyer, Ferranti Ltd., Edinburgh

Chairman: Mr. C. F. Schofield, University of London Computer Centre, London

Attendance: Day 1 (Day 2 had more people - after merging of groups)

Name	"Company"	Main Interests
S. V. Dyer	Ferranti	Numerical Control
C. F. Schofield	U.L.C.C. Univ. of London	Common sense
I. C. Cullen	B. P.	Compatibility (Mobility)
Miss S. Epittem	Cranfield Inst. of Tech.	"Standard and easy-learn" FORTRAN
H. Chattin	P.M.A. Consultants	Character handling to be standardised
R. G. Pringle	Bath Univ.	General
Dr. M. A. Hennell	Liverpool Univ.	General
E. O. Bodger	IBM Info. Services	Character handling (mainly) + no stupid restrictions + Mobility
P. D. Baird	Philips Industries	Character handling
Dr. M. Kennedy	Queen's Univ. Belfast	General
Dr. J. Larmouth	Cambridge Univ.	General
Dr. A. C. Day	Univ. Coll. London	General (?) (especially characters

(This is not a complete list - we had at least 25 on Day 2)

I should like to thank Mr. S. V. Dyer of Ferranti Ltd., for acting as secretary of the group. Without his help and moderating influence the meetings would rapidly have descended into chaos. We had a wide subject to cover in about ten hours. Topics such as I/O; FORMAT; and program unit “header lines" were not considered to be our problem. However the group felt the need to discuss such statements - often, although not always, with justification. I feel that most of the other groups‘ discussions were a subset of ours. The CHARACTER (sic) statement is a good example of this.

No formal proposals were discussed, and the wording below is mine. It was not possible to draw up formal proposals during the sessions, due to the very wide prospectus - and the varied nature of the group.

The agenda was the B.C.S. paper (Ref. 1) followed by the "white paper” issued at the meeting.

PROPOSALS (and Comments)

PR1. That two 'new' characters be added to the current standard ANSI set (X3.2/9). These two characters are ' (prime) and ; (semi-colon).

Comments

These suffer from the usual problems (like $ often printing as £, etc.) But semi-colon is worse in that many punching devices may not have it at all. Most should have prime. Note that prime should not be called "apostrophe" or quote ["]. The semi-colon may not be necessary - this depends on what we do with multi-statement lines. The use of these two characters is described below.

PR2. That two forms of "character" constants be introduced in addition to the Hollerith (H) constant. These new constants are:-

nRn characters [integer constant]

and

’n characters’

There shall be no limits specified for n except that it should (?) b greater than zero. The characters may be any of the (extended) standard set. If the prime character appears in the second form then it should/must be immediately be followed by another prime (no blanks, etc., may intervene). Such contiguous primes shall be converted to one prime character by FORTRAN processors. (Needs rewording).

[These three constants (H, R, and ') will be described as "literals" in this report].

The form ‘n characters’ shall be equivalent to the current specifics for nHn characters unless one (or more) of the characters is a prime - in which case we need a clear rule.

The form nRn characters shall produce an integer constant. Loosely speaking; an item of the form 1R character is replaced by a unique integer between 1 and a limit L.

L, and the integer mapping, may be implementation dependent. Normally, L would be a power of two, corresponding to the number of bits used to represent a character; and the mapping is the so-called "internal character code." Preferably, L should be 2**8. And the code mapping should be the ISO code mapping.

The integer corresponding to nRn characters is the integer given by "representation to base L." For example;

```
if    K=3RABC , and
      M=1RD   , then
      KM=4RABCD
```

Thus it is true that

KM=K*L+M

We can thus handle characters without ‘CHARACTER'.

If n is greater than (or equal to) the number of characters m which can be stored in one storage unit, then only the m leftmost characters will be significant, and the action taken for any remaining (n-m) characters shall be undefined.

Comments

There was a very clear majority on the requirement for (not desirability of) both new constants.

We must recommend R constants. This will be useless unless we also have R conversion in FORMAT. The correspondence of R with R is not "irregular". Maybe the correspondence of A and H is. We did not consider "A" constants - if you want to do that, the "A" should mean "H" in literals (not in FORMATs).

The double embedded prime is a problem:- it is irregular. IBM/360 (etc.) do it. If we do not like it we must say what to do with ‘’’ and ""etc.

PR3. That "P1" (Ref. 1) shall stand: but complex operands shall also be permitted (above D.P.) The meaning of (arithmetic) expressions containing literals (or ‘CHARACTER’ operands) shall be undefined unless such expressions contain no other operands or operators.

Comments

There was scant sympathy for leaving COMPLEX out due to the COMPLEX*16 problem.

PR4. That where an identifier (name) contains more than 6 characters, all characters after the 6th shall be ignored.

Comments

Unanimous from the floor - despite my strong opposition, on the grounds of CDC/1900 compatibility.

PR5. That a new type be introduced - "DOUBLE PRECISION COMPLEX". Items of this type shall be above COMPLEX in the type hierarchy, and shall occupy 4 storage units.

Comments

None (!)

PR6. (Alternative to P1/PR5). That arithmetic and relational expressions containing operands of different types shall be evaluated in the mode of the "highest" type operand. (Simple!):- X+I/J. And the expression then be evaluated from the left.

Comments

My idea - very much liked by the floor. I have been told that it is hard to implement and makes optimising hard. But it is still a nice rule. What is the PL/1 rule? Someone said that we should not go out of our way to clash with PL/1 - he has a point.

PR7. Relational Expressions

That two new relational operators be introduced:

.EQV. and .NEQ.

(or .ID. and .NI.) (etc.)

These operators shall mean "the same as" or "the same bits" - i.e. a "strict" comparison.

Comments

The problem is that (A~B) etc. can produce zero results even when A is not the same as B. This problem appears on many machines - and thus forces the compiler writer to plant risky efficient code - or safe slow code. This is a serious problem which must be resolved (unanimous). It can be resolved by the introduction of ‘CHARACTER’. Viz. - if one of the operands is of type CHARACTER then: "I want a strict (logical) comparison - don't just subtract.”

"P1" was pretty simple minded in grouping "Arithmetic and Relational Expressions" together. They are not the same. However, it would be nice if they were. See also WP9 and WP9(a), (b).

PR8. ex_1 .rel. ex_2

If neither ex_1 nor ex_2 are of type logical (or CHARACTER) nor literals, then the meaning shall be as current ANSI. [I would like to see this comparison done in the "highest mode" - with a "real"-part-only for COMPLEX and D.P.]

If ex_1(or ex_2) are literals - or expressions of type CHARACTER then a "strict" comparison shall be done.

If either ex_1 or ex_2 are of type logical then the result of the comparison shall be undefined.

If rel is one of those under PR7 then a strict comparison shall be done regardless of the forms of ex_1 and ex_2.

Problem with: (1.D2 .EQV. 1HE)--- i.e. different lengths.

PR8(a) The ALGOL "conditional expressions" were rejected by the group.

PR8(b) The SDS-invented "extended expressions" also arose. This is a real problem because it conflicts with (syntactically) our ideas for everything else. To put it simply, SDS treat "=" as an operator (of the highest order) - and they thus provide a most elegant form of multiple assignment. The SDS expressions deserve consideration.

PR9. That A**-B be accepted as A**(-B).

Comments

Unanimous (sheepwise). I do not know - except that the unary minus should be sorted out - what is -A/B or -A**B ??

PR10. Arrays may be referenced by a single subscript. E.g.

```
      REAL A(10,10,10)
      DO 2 I=1,1000
    2 A(I)=0.0
```

Comments

Widely accepted - but not on (?). Good FORTRAN II practice, but modern compilers optimise and in this case the (optimisation) means punching more source statements!

PR11. That arrays shall be permitted to have at least 7 dimensions.

Comments

Unanimous.

PR12. Another thing which had a fair amount of support was "arrays start at A(0) - not A(1)" :- as in ALG¢L.

Again, SDS allow this - as in:

```
REAL A(0:10),B(3),C(-5:3.6)
```

We do not have to go as far as SDS, but if we allow the "0" then we might as well finish the job. This is not hard to implement - but it could make optimising hard.

APR1 Problem commas and lack thereof (examples):-

1) DO 2, I=1, 10
2) WRITE (7), J
3) PRINT 1 A
4) DATA A/1/ B/2/
5) COMMON A,//B(/) [?]
6) WRITE(6)(A(I) I=1,2)
7) GOTO(1,2,3)I
8) GOTO I(1,2,3)

1) Insertion of redundant comma after DO label. Many students do it. Why don't we suggest it as an optional extra?

2) Same as (1) - but more regular(?)

3) Less important (especially if you have killed PRINT - that was a nasty thing to do) - syntax problem, so not on.

4) Unanimous - comma should be optional.

5) Same as (4) - but only I thought the comma was needed by ANSI (I hope that it is not).

6) This is like (3) - syntax problem if no comma, so probably not on.

7) Unanimous - stupid comma in computed GOTO be optional

8) Problem. Assigned GOTO comma is apparently redundant - but if left out it looks like GOTO function. Maybe FORTRAN 2000 will accept this - so not on?

9) Any similar problems?

APR2 That the (comma and) list of statement numbers shall be optional in the ASSIGNed GOTO statement.

Comments

Well received, but objections raised as regards implementation. I do not think there is an implementation problem if you are prepared to sacrifice execution time if you leave out the labels. Most FORTRAN's let you do it. Serious proposal.

APR3 That:-

REWIND(I) (etc.)

be allowed - because the brackets are required round

WRITE(I) (etc.)

Comments

Well received - for a revolutionary proposal (not mine). Advanced compilers accept this - because they accept expressions anywhere.

APR4 REWIND (etc.) exp_1, exp_2, ---- exp_n

The exp are (integer) expressions (excluding I/O function references). Some compilers allow this. There are no syntax problems.

APR5 New statement (M/C dependent? - so is PAUSE)

UNLOAD exp_1, exp_2,---- exp_n

This "gets rid of" the file (data set) for the job. It is very useful for long tape jobs. [Not discussed - by us].

BPR1 That (often 'integer') expressions may appear wherever they make sense. Viz. in:

1) Output lists (excluding I/O function references) - (any type)

2) As DO and implied DO parameters (excluding I/O functions)

3) As Computed GOTO controls

4) As "device" numbers in I/O and REWIND/UNLOAD etc statements (controversial)

5) As subscripts.

6) As adjustable array dimensions.

7) Anymore?

CPR1 That "P2" shall stand (forgetting type DOUB.PREC.COMP. - good names are DUPLEX or COUPLE). We also want to permit literals (or CHAR. expressions) to appear on the r.h.s. This was unanimous.

Many compilers do it this way:-

If the r.h.s. is a literal (or a CHAR.) then the r.h.s. shall be stored in the l.h.s. "variable" without conversion of any kind. If the literal (etc.) is too long to fit into the l.h.s. then only the m leftmost characters shall be stored in the l.h.s. If the r.h.s. is 'shorter' than the l.h.s. then it shall be stored in the l.h.s. left justified with blank fill - unless it (i.e. the r.h.s.) is an R constant in which case it will be stored right justified with zero fill.

Comments

I do not suggest that wording as the standard. If the l.h.s. is D.P. or COMP, then use of literals on the r.h.s. should probably print a polite but instructive message. But maybe we should allow text to be put into double length (etc.) items?

CPR2 That a multiple replacement statement be permitted. We all wanted a standard on this. The trouble is that the idea we came up with conflicts with the SDS extended expressions. (See PR8(b)) What we suggested was:

lhs=rhs

where rhs has the meaning of ASA+CRP1; BUT lhs is a standard (ANSI) input list (as in a READ statement). This is very elegant - and probably not hard to implement - but it conflicts with many compilers, e.g, CDC which treat A as A(1,1,1). Maybe the standard ASA input list should be permitted to have our extended form (parameters may be expressions).

This goes some way to the popular request for matrix handling - in that

```
REAL A(10, 10, 10)
 :
 :
A=0
 :
 :
```

would (in execution) set all elements of A to 0. Similarly:

```
K,((A(I,J),I=1,J),J=1,K) = 35.7+SIN(Z)
```

(a bad example but gives an idea of the power). Even so, if we can do this sort of thing with the SDS extended expression then I will opt for SDS. Maybe we can do both by making an unsubscripted array name mean the whole array in SDS expressions.

That is all we had to say on replacement statements. This is an area where we can really contribute something. Someone also suggested that with an I/P list on the l.h.s¢ we should have an O/P list on the r.h.s.

PQ1 Computed GOTO ("P3")

That P3 shall stand. And that the comma be optional.

Comments

This was generally accepted (much to my disgust). My view was that a Computed GOTO error should kill the program. Two people agreed (strongly). We all want a 'default' action. The Ref. 1 'status' comment is (a) wrong, and (b) describes a bug as a "facility." As for the "rationale" it is as easy to give an error as it is to CONTINUE (in terms of planted code), We had a clear division between University and commercial users here. The latter stated that they do not have errors with their GOTO's. They could be wrong - and not know it. If P3 stands then they will never know it. Do we want to give users a (false) sense of security, or do we want to indicate what is a very serious programming error?

PQ1(a) That the Computed GOTO control 'variable' shall be an integer expression.

Comments

Unanimous. Maybe no problems unless the expression references a system function (e.g. for I/O) - is this a problem?

From here on we should bear in mind that "expression" may perhaps also mean SDS extended expressions (if we accept them) - this may involve new problems since they are also replacement statements.

Clear problem if SDS expressions accepted in DO, e.g.:-

```
DO 2 I=J,J=3+(K=7), I=6
```

We can see what SDS do with this.

PQ2 DATA

That "P4" be accepted - unanimous. Also that implied DO loops be accepted in the variable list. I said 'to what depth' = someone said 50 - I said 3. So 3 it was (unanimous). But 3 is a funny limit if we are accepting 7 sub-scripts. Depth 3 lets you initialize "sparse" matrices - so perhaps it is enough.

Also that the implied DO parameters be positive integer constants with m_1.LE.m_2.

PQ3 DO

We agreed on "P5" - but felt that it did not go far enough. Use of the word "must" is P5 should be replaced by "the action shall be undefined" - if you want to say anything (ASA usually manage to say nothing - i.e. to impose no restrictions).

PQ3(a) Real type index and parameters in DO was proposed - but had little support Why not? I hope the Americans are less conservative.

PQ3(b) So to the problem area. At this point I should explain that we took DO to mean "implied DO" as well - in the latter case the parameter-expressions should not contain any I/O system function references. We will have to define what we mean by that (like no functions at all??). The problem area is "P6" and "negative step.”

We decided that P6 might be irrelevant - but we were not sure. Given a negative step P6 (as worded) is not on.

Negative step (m3) in DO is a natural and useful extension, but it makes optimising hard. BUT a negative step may only occur if it is specified as an expression which is not a positive integer constant (ANSI).

E.g. DO 2 I=1,2,J

So "J" is the problem - is it "+" or "-"? We said: "Who cares: so few people write variable steps that it does not matter whether they get optimised or not." This is true and we can therefore get rid of the "syntactical device" suggestion. If the step is variable then the compiler writer must be_ready for anything.

So the difficult problem has been solved. Back to "P6" - or a revised version of it. The question is whether we allow the parameters (m_1,m_2) to be negative or zero - regardless of the step (m_3).

The group was strongly in favour of removing all restrictions. But I pointed out that there could be implementation problems especially if the index passed through zero. Nobody agreed with this.

I cannot resolve this m_1/m_2 problem (if there is a problem). But as a user I should like to see no restrictions on m_1,m_2,m_3. (I also want to see a real type index).

All of that applies to implied DO (and replacement statements!) too.

PQ4 EQUIVALENCE

We agreed with “P7".

PQ4(a) The mention of "mathematical equivalence" on Page 59 of Ref. 1 was not understood by us.

We decided it meant the "Forced Store Problem". There are 3 (at least) classic examples of this:-

```
1)    EQUIV(A,B)
      A=2
      C=B
```

Optimising compilers often fail on this. They put 2(A) into a register and forget that B is the same thing - thus giving rubbish for C=B. I (and many others) thought that ANSI had dealt with this - but other people thought they had not. Anyway, we said that C=B should be the same as C=A (i.e. "do it right"). If ANSI have not covered this then we should tell them to do so.

```
2)    CALL X(A,A)
       :
       :
      SUBR X(A,B)
      A=2
      C=B
```

(same problem). Do ANSI allow CALL X(A,A) ??

```
3)    COMMON A
      CALL X(A)
       :
       :
      SUBR X(A)
      COMMON B
      A=2
      C=B
```

(same problem). Do ANSI allow COMMON A, CALL X(A) ??

We cannot do much about these (but the group thought we could). Example (1) at least should be fixed.

PQ5 (Personal comment) "Gw" conversion should be allowed too. (Not my group). (“P8”).

PQ6 I hope we have agreed on "P11" - but the wording is unfortunate: "shall not" etc., should be "undefined." R constants/Formats are required.

PQ7 IMPLICIT

"P12" accepted - unanimous. None of the IBM *n stuff wanted. Someone wanted the PL/1 rule where not only the first character is significant. I do not care about that much but we should word the proposal such that it welcomes PL/1 like extensions. I suspect the next IBM FORTRAN might look a lot like PL/1.

PQ8 (Personal comment) "P13" is out. It violates your most important criterion.

PQ9 (Personal comment) PLEASE allow ERR as well as END They both interact with the operating system. (“P14").

PQ10 SUBSCRIPTS

"P15" accepted - unanimous. But I am worried about functions in expressions - we ought to clarify this situation - e.g. order of evaluation/side effects, etc.

We also want at least 7 subscripts. (This should be in the "PR" group).

PQ11 "P16" - accepted (unanimous).

PQ12 ENTRY

"S3" - accepted. But we did not think about it. No one seems to want ENTRY, but Mr. Muxworthy was right when he said that people underrate it. We want ENTRY anyway. Maybe there is an OVERLAY problem (see /360 FORTRAN manual).

"Further Suggestions”

PS1 a) Yes - see CPR1.

b) Yes - same as (a).

C) Yes (more or less) - see replacement statements, CPR2.

d), (e), (f), (g) Yes - dealt with above.

PS2 a) Yes (DATA) see above

b) (DATA) NO. What do you mean??

PS3 c) Slight regularity problem, but answer was YES.

The problem is that "X(3)" does not mean the same thing in

```
REAL X(3)/5*1.0/
```

as in

```
DATA X(3)/1.0/
```

This is a /360 idea which is gaining wide acceptance. We wanted it.

PS4 We left functions till later (much later). DINT, NINT, and CONST were wanted - and "Generic" transformation of routines, including automatic changing of variable type, constants, and function names. Great idea.

Also minimum standard of accuracy for REALs (see also CONST). You can see why I left it till later.

So to the "white paper”:-

Many people had no idea what the W.P. suggestions meant - so I put forward my own interpretations. I hope that my ideas were what you meant.

WP1 "DO extensions”

See PQ3/a/b.

WP2 "Other loops”

Not wanted. (Unanimous)

WP3 "Multiple replacement”

See CPR2

WP4 "Matrix arithmetic”

Covered above - to an extent (see CPR2) ~ but we did not want to go mad about it. We should not propose anything which conflicts with current PL/1, e.g. A(*,3,*)=0

Functions not discussed.

WP5 Computed GOTO

See PQ1, PQ1(a)

WP6 ‘Blank labels in IF/GOTO’

Yes - we want them. General preference for zero (0) rather than null labels - which produce ugly statements. [Alternative suggestion was * for "next statement" - with things like 5* in Computed GOTO. I think * should be rejected.]

Problem raised was:

```
ASSIGN 0 TO I
  :
  :
GOTO I
```

In this case the GOTO I should mean CONTINUE (if it is to be defined). with null labels the ASSIGN statement would be

```
ASSIGN TO I
```

I do not feel that to be a problem, but still prefer "0" to "null". Either way we want this extension.

WP? "Hollerith in replacement”

See CPR1

WP8 "ELSE in logical IF”

Not wanted, but mainly because I deferred it to WP1O. A fair amount of interest was expressed.

WP9 "Multiple IF" (etc.)

We took this to mean

```
IF(A=B)IF(C=D) GOTO 3
```

Since this can be written

```
IF(A=B.AND.C=D) GOTO 3
```

the idea seemed odd. We rejected it. Abbreviations .A., .O., .N. wanted.

WP9(a) Notice my use of "=" for ".EQ." in the above IF. We wanted to allow for this sort of extension. This is why we went for

```
A,B,C=0
```

rather than

```
A=B=C=0
```

Since the use of "=" for '.EQ.' conflicts with the multiple logical replacement statement in the second form. (This is how we finished up with an I/P list on the l.h.s.) Other syntax problems? (Yes - if we have SDS expressions).

WP9(b) We also want things like

```
IF(A.EQ.B.EQ.C.LT.D) --------
```

No problems. Unanimous.

WP10 "Compound IF”

This was definitely wanted by all - but was very controversial. I

described the Atlas FORTRAN V way (and SDS) ~ which is the IF followed by a multi-statement "line" (i.e. statements separated by $ [or semi-colon] - covering 20 cards if required). No labels may appear after a $ - so no jumps can be made into the compound statement. The compound statement should not contain DO or IF statements - but GOTO's (all kinds) are okay (zero labels mean next “$").

This assumes that we accept multi-statement lines. We should (there was not much argument about that). If we use $ as a separator we will please CDC, but clobber /360. This is a good case for clobbering IBM - the idea of allowing $ in names is the craziest action I have yet heard. However we can still use ";" (like SDS).

People thought that I was talking about ALGOL block structure and this tended to confuse the issue. So many ideas and objections were raised that I had to call a halt. One idea that I do remember was that we follow the IF with a DO - thus not needing multi-statement cards.

```
E.g.              IF(A>B) DO 2 I=1,10
                             :
                             :
                          etc.

                  2 CONTINUE
```

This appeared from the PL/1 expert (as did many of the best ideas). At first sight it looks peculiar - but the more I look, the more reasonable it seems. I like it - it is a superset of the FORTRAN V idea.

The ELSE idea was raised again, but I had had enough new ideas for one day. Whatever we do (and we must do something) I think that we should forget about ELSE.

The FORTRAN V/SDS method has been found very acceptable by users and is not hard to "read". Most people thought it would lead to a mass of $ statements which would be unreadable. This is not true. The pedant can punch one statement (followed by a $) per card - and still get a 20 statement compound IF.

WP11 "Extended Statement Functions”

The short answer was NO. But I took this to mean the UNIVAC DEFINE/ PARAMETER (another new thing) - which nobody knew. We ought to look at the UNIVAC ideas on this - they are powerful.

WP12 "Conditional Expressions (ALGOL)"

See PR8(a)

WP13 "= as operator"

See PR8(b)

WP14 Sort out Standard Functions

Yes we ought to -

e.g. ATAN 2 and IFIX/INT

But we did not discuss functions (no time).

WP15 As WP14 (distinction is invidious).

WP16 GENERIC

Yes - very powerful idea and I would think not hard to do - see earlier comments. (PS4).

WP17 DINT and NINT wanted - but no time for discussion of the many others which would have been suggested.

WP18 "EQUIVALENCE extensions”

No. What do you mean? (EQUIV is bad enough already).

WP19 "Data in Type Statements”

See PS3.

WP20 "DATA extensions"

See PQ2.

Other ideas

QP1 Allow assigned label variables to appear in IF/GOTO (and CALL?) statements, E.g.

IF(I-J) M,N,3

Nice idea.

QP2 Argument-driven functions e.g, SQRT(2.D0) means DSQHK2.D0)

Yes. (Does DSQRT(2.) mean SQRT(2.) ??)

QP3 PUBLIC (or GLOBAL)

Wanted - unanimous. (Involves loader).

Very useful - better than COMMON.

QP4 (My idea). RETURN shall be allowed in main program - it shall mean "return control to the operating system - or STOP if you do not have one." I do not care what happens, but RETURN in main program should NOT be a fatal syntax error.

QP5 That we suggest RETURN i. We should. Few people wanted it. The IBM ways are poor. The CDC FTN (Extended) way is good. I suggest the latter be a formal proposal.

OP6 (Clive's favourite function - so be nice to it).

We can argue about the name and the form of the arguments. This function is entirely M/C dependent - and that is the whole idea. It is trivial to implement, but hard to specify. The idea is a bit like ‘Generic’ - i.e. m/c independent programming.

I will call it CONST - and it will have one text (H or prime) argument. E.g.

```
I=CONST('CHARS')
```

gives (in I) the no. of characters which can be held in 1 storage unit.

```
I=CONST('MAXINT')
```

gives biggest integer.

```
X=CONST('PI')
```

gives PI to full real accuracy.

```
D=CONST('DPI')
```

gives PI to full d.p. accuracy.

```
I=CONST('ACCREAL')
```

gives accuracy (decimal digits) of real numbers - and so on.

Useful in so many ways - E.g. for the many programs which need CDC double precision (29 digits):-

```
IF(CONSTC(ACCDP).LE.20) STOP
```

would save a lot of time on other machines (like /360).

This function (or functions) is unusual in that we need to specify the arguments as well as the function name (but we cannot specify the results!) I would very much like to see a proposal on this - we do not have to be too formal about it. The idea was liked by the people that understood it.

And that is all we discussed, (I think).

C. F. Schofield
Group Leader Compilers
U. L. C. C.

23rd April 1971
/GED

Executable & Specification Statements

P1: The following points received general support

i) that type complex should appear in the hierarchy of types

ii) that type double precision complex should be included in the standard

iii) that expressions of mixed mode should be entirely evaluated in the mode of the operand highest in the hierarchy contained in the expression

iv) that the special case of contiguous operators **- should be permitted

v) An array shall be referable to with single subscript.
It was realised that this was crying for the moon.

P2: This was unanimously agreed with, subject to above comments on Pl where relevant

P3: There was unanimous agreement on the need for standard default action.

Two strong personal views were expressed in favour of this being a fatal error condition.

P4: This was accepted, but there was a strong feeling that implicit D0-loops should be permitted, with nest depth of 3, and positive integer constants only.

P5: Additional proposal is to remove all restrictions on ml, m2, m3.

P6: Accepted since it is a de facto standard, but only under strong objection.

P7: Accepted.

P12: Accepted.

P15: Accepted.
But some concern was expressed at what ought to be left when this proposal is found not wholly acceptable.

(Problem area is subscripted arrays in subscript expressions for optimising compilers).

P16: Accepted.

S3: Accepted

Further suggestions

a) Yes.
b) Yes with "input" lists on LHS.
c) Solved by b).
d) & e) covered.
f) Opinion divided, but consensus to keep comma.
g) Yes please.

[Editor's note: This is a transcript of the document as circulated. The interpretation of the items is not now clear.]

General Expressions in Output Lists

Proposal 15 in the Computer Bulletin already copes with some of the problems. Array names should not be permitted in expressions. Functions should not perform I/O. There was some doubt concerning the need for this extension.

T format

Changes to numeric format codes (so as not to overwrite with blanks) was felt not to be desirable.

R format

R format introduces problems of word length. It is very machine dependent. Furthermore, there is a lack of a data statement to correspond with it. This format code is not needed provided that type CHARACTER is introduced with some means of transforming single characters into small integers. Fear was expressed that neither the format code nor the facility concerning characters might be provided.

Reread and ENCODE/DECODE

Reread implies reading again the last record read, using a different format. ENCODE/ DECODE implies reading (or writing) from (or to) a program array. Implementations are mostly of the ENCODE/DECODE type. This should be added to the standard as READ and WRITE with an array name in place of the unit number. (Or possibly a character variable or array as in WATFIV.) It was decided (with some dissension) that reread was needed as well. This should have the keyword REREAD and the remainder of the statement as for READ.

Format code/ and rescan are forbidden.

Free format

More the province of the conversational committee. This is needed, but there are so many different ways in which it is being implemented at the moment.

Namelist

This is good for debugging, and many compilers provide it.

READ, PRINT, PUNCH

Many programs still use them, but the standard should not include these facilities. A variable can be used for the logical unit number.

Random access

The current implementations tend to be ad hoc botches, and result in that which is not FORTRAN. Some method is needed.

Buffer in and out

Some operating systems buffer all I/O at the moment. Buffer in and out was rejected.

Additional Types (Proposal for type CHARACTER)

Declaration e.g.

```
CHARACTER* 7 A(5), B, c
```

means B can hold 7 characters, A is an array of 5 elements, each of which can hold 7 characters.

If the *n is omitted, *1 is assumed,

Assignment

$char_1 = char_2$

where $char_1$ and $char_2$ are character variables (subscripted or not). If $char_2$ is shorter than $char_1$, the remainder of $char_1$ is filled with blanks. If $char_2$ is longer, the leftmost characters are assigned, $Char_2$ may also be a Hollerith constant or a character function reference,

DATA statement

```
DATA B /5HABCDE /, C /'l23'/ .
```

Constants shorter than variables are padded to the right with blanks.

I/O

Reading and writing can be performed using A format code, One variable corresponds to one A descriptor, If the length of the descriptor does not equal the length of the variable, the same action is taken as at the moment.

Core-to-core READ and WRITE

```
READ(c,f) list
WRITE(c,f) list
```

where c is a variable or array of type CHARACTER.

EQUIVALENCE

Character arrays and variables may be equivalenced together in any way which does not redefine the origin of a COMMON block. Characters placed in a variable of type character are also placed in the corresponding parts of other variables of type character equivalenced to it.

FUNCTIONS

Character functions take the form:

```
CHARACTER*n FUNCTION name(pars)
.
.
.

END
```

Library functions

External functions should be provided (1) to transform a CHARACTER*1 variable or constant to an integer which is the ANSI representation for that character and (2) to transform an integer into a CHARACTER*l result.

Constants

Hollerith constants are the constants corresponding to type CHARACTER.

Character*l

WATFIV

if lengths are unequal and if assignment is to the longer then the remaining space is filled with spaces, but if assigned to the shorter then truncate.

```
Read(C,lOO)
```

Character Function creates no problem.

Mixed mode expression should be forbidden.

Character*l causes some inefficiency. Need to access substrings. Comparison of strings of different lengths imply the shorter is expanded to the right with spaces before the comparison is made.

One should be able to equivalence variables of different types (one of them a character variable).

System Function to obtain internal code representation of a particular character.
eg. J = ITOC(lHl)

Objections to WATFIV	Strength
Character can be used as a subscript if C(i) = 1 then A(C(i)) => A(241) if 1 has internal code 241	Substrings Ability to assign Ability to ENCODE/DECODE -> A format

Need for functions

If J = 24l then C = Character l
C = CTOI(J)

Need for Character rather than existing H & A format. Is inefficient coding necessary to implement?

Certain machines will not have all of the character set. There is a need for a default setting.

1900	does not have equivalent to IBM 360
CLC	instruction for comparing character strings

[Editor’s note: This page has been transcribed as distributed, although the meaning remains obscure.]

High level language should be removed from bit patterns. Fortran should not be related to a particular machine.

Proposal from Colin Day

Character{*n} n may be a non zero positive unsigned integer.

Assignment statement

$c_1 = c_2$

where c_2 is another character which may be of different length.

If $c_1 > c_2$ then c_1 is right filled with spaces

if $c_1 < c_2$ it is truncated to length of c_2 and c is declared as Character*n
C(80), A(5)

```
            EQUIVALENCE (A5),C(75))
            Read (C,l5)I,J,R
15          FORMAT(3Il0)
```

This may be complicated to implement on word machines.

```
      I = IANSI(C) This will convert character
      C = CANSI(I) to ANSI value and vice versa
```

where C must be Character*1

```
Character*3 C(8))
Character*2 A(5)
Equivalence (A(2), C(2))

                          C(1)|C(2)|
                              |    |
means if C contains        ABC|DEF |GHI
```

Therefore A(2) contains DE

IF was dropped due to lack of agreement, although it was thought essential.

FREE FORMAT LAYOUT OF SOURCE CODE

The Free Format Layout of source code is necessary for

(a) small computers whose main input media is paper tape
(b) Time-sharing terminals

and desirable for large computers with a choice of input media.

Recommendations

(1) That the new standard should include a specification for free format layout of source code and that this facility be part of the standard and not optional. The layout should be truly free, viz. Spaces will be ignored everywhere except in Hollerith Strings, i.e. "column 1" for comments and "column 6" for continuations have no significance (see also recommendations 3 and 4).

Free format layout must be provided on all forms of input (i.e. including cards).

(2) That the 'end of line' be defined as

(a) a special character defined by the Operating System e.g. 'new-line' on paper tape.
(b) occurring immediately after the last non-blank character preceding column 72 on cards.

(3) COMMENTS: A special character as the first non-blank character in a line. Characters would be ignored until the end-of-line character.

(4) CONTINUATION LINES: A special character followed by an end of line character specifies that the next line is a continuation line.

(5) To allow more than one statement per line, the end of statement be defined as

(a) end of line
(b) a special character

N.B. in recommendations 3, 4, 5 the special character must be specified in the Standard and should be $ or a character not already in the FORTRAN character set. These three special characters need not be different, e.g. the comment character and the continuation character could both be $ and the end of statement be ; .

(6) Additions to the Character Set

(a) Tab character - to be interpreted as one or more spaces

(b) New-line character

(c) Any special characters required to provide recommendations 3, 4 and 5.

(7) That the mixing of free and fixed format layout be allowed. Thus two directives, say FIXED and FREE, be provided to switch mode.

The current mode will remain in force until either FREE, FIXED or end of full program occurs. Two successive FREE or FIXED directives will not

(7) Con/...

not produce an error condition.

The default mode need not be standardised but a default must be provided.

D.H. MARWICK.
(Chairman)

INPUT

Recommendations

(1) That the type of the item to be read is taken from the input list.

(2) That the format allowable for each type of item be as in Table 1.

(3) That the item terminator as in Table 1 is not ignored i.e. it is considered as part of the following item.

(4) That the statement indicating free format input be

READ (u,) input list

where u is the input unit

(The merits of

READ (u, o) input list

were also stressed both in the group and in general discussion, though it was felt that this should be compatible with the solution to similar problems of missing labels in the Arithmetic IF and the Computed GOTO)

NB: It was also recommended to compiler writers that the statement

READ (u,f) input list

be treated as a free format READ statement if the statement 'f' is missing. This is felt to be desirable for mini-FORTRAN and so should be included for upward compatibility.

(5) That free format input is not record based, but item based. Thus a new READ statement causes the next item to be input even if this is in the middle of what is considered to be a record in fixed format input.

NB While the group made no recommendation on the mixing of free and fixed format READ statements, it was strongly recommended that this point be explicitly specified in the Standard.

TABLE 1

TYPE	FORMAT	CHARACTERS TO BE IGNORED		TERMINATOR
		IN ITEM	BEFORE ITEM	
Integer	± integer constant (+ is optional)	None	space, new line, tab	any character which is not part of the integer const. which must consist of at leat 1 digit
Real	± integer constant ± real constant (+ is optional, Decimal exponent is E or D)	Spaces after the decimal exponent	space, new line, tab	any character which is not part of the real constant, which must consist of at leat one digit in the mantissa and, if there is a decimal exponent, at leat one digit in the exponent
Double Precision	as for real	as for real	as for real	as for real
Complex	a pair of Real items	as for a pair of real items	as for a pair of real items	as for a pair of real items
Logical	first character must be T or F	any alphabetic character	space, new line, tab	any non-alphabetic character
character (CHAR*n)	any n characters	none	none	self-terminating

[In the original typescript the table was presented in landscape mode]

OUTPUT

Recommendations

(6) That the items Output under free format must be capable of being re-input under free format.

(7) That the field width of each item should be a multiple of x characters. The total field width will be the smallest width able to contain the item to be output.

(8) That each item will be preceded and followed by a space, which must be included in the field width, and right justified in the field.

(9) That each type of item is output as follows:

Integer - to full accuracy with sign if negative
Real and Double Precision - to y significant figures, i.e. G W. y
where w is the smallest multiple of x possible (see 8)
Complex - as two Real items
Logical - T or F in a field width of x characters
Character (CHAR*n) - as the complete string, i.e. in a field of width

INT ((n+x+2)/x) * x

where INT(a/b) is the largest integer less than a/b.

(10) That the values of x and y be set as standard or be specified by the user calling a standard subroutine, say, IOPARS(IX,IY)
e.g. CALL IOPARS (5,6)

(11) That the statement indicating free format output be the same as the free format input statement with WRITE instead of READ (see recommendation 4)

(12) That free format output is not record based.
Thus, if there is room on the current line for the next item, it should be output on that line, i.e.

(a) a new WRITE statement does not start to output on a new line

(b) a new line character does not split an item (see recommendations 6 and 2)

Chairman's Comments

It became very obvious in the group and in the general discussion that free format I/O of data has a different meaning depending on the person and/or the application. I think there is a case for considering free format under more than one heading and proposing standards under each heading

e.g. (a) Simple facilities - where the application will be as a debugging aid or a simple I/O facility for mini-FORTRAN and educational purposes.

(b) As a true alternative to Fixed Format - where fairly complex facilities are required but details of field widths etc are not. This facility would require new statements.

David H. Marwick
(Chairman)

PROGRAM STRUCTURE: SUMMARY OF DISCUSSION 6.4.71

1. Adjustable dimensions passed in COMMON.
   This facility is not strictly necessary and may put a constraint on implementors, particularly if the code is re-entrant. It was not recommended.

2. Differing numbers of arguments in call and subprogram.
   There should be an option to allow for differing numbers of arguments in call and subprogram. Further, with a view to linking other languages, the implementor should cause the number of arguments at a subprogram call to be available, and the number of characters in a Hollerith constant argument to be available.

3. Call by name and call by value.
   The group deplored the 360's copy-in-and-copy-back rule and wished the current practice - passing an address - to be formalized.

4. RETURN i
   This was not felt to be necessary.

5. Initialization of variables.
   Variables should continue to be undefined at the start of execution of a program.

6. ENTRY
   The group supported the feature already proposed (in the Bulletin) and wished it to be extended to conform with current practice, using the 360 implementation, and not the 6600 as a guide. ENTRY should also be available in a FUNCTION.

7. GLOBAL-PUBLIC
   This feature was not thought to be necessary.

8. Block structure and recursive calls.
   These features were thought to be too fundamental a change and could not be recommended.

9. Overlay and local variables.
   There was a desire to define overlay and definition of local variables so as to increase the portability of programs between machines. Discussion revealed that there was more variation amongst existing implementations than had been realized and that the matter needed deeper discussion.

The only definite suggestion was that entities in blank common should be permanently defined.

10. Dynamic Arrays.
The question of dynamic arrays was discussed. It was thought be a most useful feature which could not be programmed around. There was no agreement on the point at which the array size could be decided, or whether this item could be forwarded to ANSI.

11. Compiler Electives.
The question of whether the standard should allow electives, as in COBOL, was forwarded for more general discussion.

12. Extended Core.
If some variables are to be kept in a different store from the rest of the data (e.g. extended core, LCS)(and this must be known at compile time) the group deplored the introduction of new type statements and recommended that such variables should be put in a labelled common block with a special name.

13. Compiler Options.
Should options which are generally available - such as whether a listing is to be printed or whether a binary deck is to be punched - be put in a standard form, and should this be part of the Fortran language?

14. EXTERNAL
It was suggested that the effect of the EXTERNAL statement should be classified.

15. Other items.
There was discussion, but no positive recommendation, on correspondence of argument types at a subprogram call and on the scope of an ASSIGN'ed variables.

DIAGNOSTICS AND PROGRAM STRUCTURE: SUMMARY OF DISCUSSION 7.4.71

1. Compile-time diagnostics

There was considerable discussion on these points:
- what is a fatal diagnostic?
- should it be possible to switch off diagnostics?
- should there be a minimum standard set of diagnostics?

Agreement was reached on these points, which do not necessarily affect the Standard:
- that it should always be possible to have printed a list of the variables used in a program unit,
- that manufacturers should list diagnostics and causes in their documentation,
- that a comparative study of diagnostics would be most welcome.

2. Run-time diagnostics

Discussion ranged over:
- should array bound checking be non-existent, optional or compulsory?
- should emphasis be on debugging or production?
- what should be the action at an error - jump to standard label, standard subroutine, main program and how can one (sensibly) recover control?

Agreement was reached on:
- any run time diagnostic should include at least a trace-back (360 terminology)
- there should be standard functions for time and date and the time left for this run
- that ERR= as currently implemented should be available, so that at least a faulty record may be skipped
- that at an error the user should be able to retain control, if only to skip to the next set of data and restart.
- that introduction of the PL/I "ON" condition would not fulfil Fortran user's needs.

Most importantly the group recommended that a working party be set up to discuss compile-time and run-time diagnostics more thoroughly.

3. Random-access I/O

Four conflicting views were put forward and were not resolved:

- the IBM implementation has been widely copied and should be standardized,
- that no new statements are necessary as the effect can be achieved by subroutine calls,
- that some standard should be recommended to avoid chaos,
- that no standard should be recommended as it may restrict new hardware.

4. Multi-programming

There should be a statement or standard subroutine which indicated to the operator that a device was finished with.

5. Data sets

A Fortran compiler must be such that the logical units are device-independent at compile time. Further, the Standard should specify a minimum number (say 8) of I/O devices which can be open at any one time.

6. Overlay

It should not be necessary for the Fortran programmer to write explicit overlay statements in the body of a program. Common blocks and library subroutine should be automatically placed as low as possible in an overlay tree.

It was desirable to bring the Standard's attitude to local variables more into line with current practice and this may be possible by defining a segment in the Standard.

7. Standard options, debugging, comments

It was suggested that a set of standard options (e.g. source listing, object listing) should be specifiable in a standard format. Further it was suggested that standard debugging aids be available. (Chairman's comment: It appears that cards beginning C$.... are gaining general acceptance, see e.g. Computers & Automation February 1971). The group recommended against having comments on statement lines.

D.T.Muxworthy
(Chairman)

CONVERSATIONAL

LAYOUT - of program statements

1. Line numbers essential for purposes of editing and diagnostic reference.

2. Specify space after line number, followed by the statement. (overridden for comments and continuation lines.)

3. CONTINUATION - special char terminating preceding line. System should then replace post-line number delimiter by char signifying continuation. This was not a unanimous proposal.

4. COMMENT - no agreement.

5. INTERACTIVE COMPILERS
Editors - text versus line some confusion as to the current ‘de facto' standard and trends.
Line numbers should not be confused with statement numbers.

DIAGNOSTICS

Uninitialised variables should be recognised by specific bit pattern.

Much discussion took place on what constituted a conversational system.

MINI-COMPUTERS

1. Consider only machines with more than 8K bytes, (4K-l6 bit words) in the future will anybody want Fortran on smaller machines?

2. Require a well defined subset that is upward compatible, and defined so as to be sideways compatible? (Tomorrow).

3. The new Fortran standard should allow free format for Source code statements to allow compatibility between non-card users. (Tomorrow).

4. Interrupts or not! Hardware interrupts
   a. Not expecting Software interrupts (div by zero)
   b. Waiting for interrupt
   c. Output under interrupts (Tomorrow).

5. READ(N,L) list or WRITE(N,L) list
   If no format statement with that label then default input/output.
   Input take next input field until first delimiter.
   ignore leading blanks, unspecified delimiter.
   Output must be capable of being reinput under free format, possibly to full accuracy.

B.C.S. Fortran Specialist Group

Small Computers

At the Edinburgh Meeting in April the following people attended the meeting of the working party and expressed interest in continuing the work in this field.

M.J. GARSIDE	Computing Laboratory, Cornwallis Building, The University, Canterbury. Canterbury 66822
I. Davidson D.W. White	UKRSC, National Cash Register Co., 206 Marylebone Road, London N.W.l.
A.S. Jordan	Civil Service Dept, Computers Division, Richmond Terrace, Whitehall, London S.W.l.
C. Lemming	Marconi Elliott Computer Systems Ltd, Borehamwood.
M.J.D. Moway	D.A.F.S. Marine Laboratory Victoria Road, Torry, Aberdeen.
Mrs J. Muscott	A.R.C. Unit of Statistics, University of Edinburgh, 21 Buccleuch Place, Edinburgh.
A.R. Sibbald	Hill Farming Research Organisation, 29 Lauder Road, Edinburgh.
D. Winstanley	M.J. Bevans Ltd, 40/42 Washway Road, Sale, Cheshire. (Home 06l-962-92l6)
B. Shearing	Alcock Shearing & Partners, 8 Iddesleigh House, Caxton Street, London S.W.l.

Report on the Edinburgh Meeting

The group had a certain difficulty in identifying what was a "Small Computer". Some members thought that it was unrealistic to expect Fortran on a machine of less than 8K bytes. Others thought that it might be possible to use Fortran as a language for smaller machines if the programs were to be compiled on a larger service machine.

The working party considered the available versions of Fortran on small machines and noted that there was a considerable variation in the facilities available with machines such as the Honeywell DDP5l6 appearing to offer almost full ASA Fortran. This would seem to be a guide line as to what can be done on small machines.

The working party identified several problems that need further study and expressed the opinion that these problems were to a large extent caused by inadequacies in the existing Fortran language definition. They therefore felt any revision of ASA Fortran should take into account the needs and problems of the small computer user.

1. Fortran Character Set

The ASA Standard lists 47 characters as being in the Fortran character set. This led to various contortions such as .LT. for <. Various paper tape based Fortran systems (re PDP-8) have extended the character set to allow the use of symbols such as <. This has introduced considerable lack of compatibility which should be rectified by an extension of the standard character set.

2. Paper Tape Fortran

ASA Fortran defines a line to be 72 characters with the statement in positions 7 thru 72. The growing use of paper tape based systems (on line and off line) has led to local divergencies primarily in regard to a departure from the convention of commencing statements in position 7. Following Algol the symbol <;> is used to terminate statements in multi-statement lines. The working party is not at present putting forward concrete proposals but is pointing out the inadequacy of the Standard to deal with a non card/batch based system.

3. Input/Output

There appears to be a need in the paper tape environment for a format-free input. This would have an advantage in the small machine environment in that there would be no need to hold a large run-time input/output package. The working party is not putting forward definite proposals but would like to see some form of delimiter (such as ,) used to terminate input fields. The language definition should contain some facility (such as a reference to a non-existing Format statement or to label 0) to indicate format-free input.

A similar language facility should be provided for output in which case default printing to some pre-determined format would result.

Restrictions

The working party felt that a restricted defined subset of Standard Fortran might be desirable on a small machine so long as this subset was upward compatible. It was thought that some restriction on the ordering of Declarations would be acceptable if this was to reduce the size of the compiler. The EQUIVALENCE statement seems to be of doubtful use on a small machine and the DATA statement used in sub-programs is of doubtful value unless linked to the Algol concept of own variable.

M.J. Garside
16th June, 1971.

Minutes of the final session of the Fortran Workshop, held in the William Robertson Building George Square, Edinburgh on Wednesday 7th April, 197l at 4.30 p.m.

1. OFFICE-HOLDERS — The Specialist Group Chairman announced that Mrs Barritt had resigned as Group secretary and that it had been arranged that Mr D.T. Muxworthy would be minutes and meetings secretary and Mr R.E. Day documents secretary.

2. WORKING PARTIES — It was decided to set up four working parties:

	subject	chairman	to meet in
1.	Free Format	D.H. Marwick	Edinburgh
2.	Mini-computers	M.J. Garside	(not fixed)
3.	Diagnostics	P.A. Samet	London
4.	Extensions	B.H. Shearing	London

Of these numbers 2 and 4 are essentially continuations of existing working parties and numbers l and 3 are new ones.

3. TRENDS — Mrs Barritt urged that a consensus of opinion on future trends, not necessarily currently acceptable as standards, should be communicated to ANSI.

4. WORKSHOP REPORTS — Mr Gatehouse asked all discussion group chairmen to submit reports on their discussions to the secretary. The Steering Committee were to edit them and produce a report for the Computer Bulletin or Journal.

5. CONCLUSION — Mr Gatehouse conveyed apologies for absence from the Secretary-General of the Society, and thanked the Edinburgh Branch and E.R.C.C. and particularly Mr R.E. Day for their organization of the Workshop. This was a new example of cooperation between a Specialist Group and a Branch. Mr Gatehouse asked the participants to remember the average user when suggesting changes to the language. On behalf of the Edinburgh Branch, Professor Balfour thanked the participants and Mr Day for making the workshop such a success