2nd AFSC
STANDARDIZATION
CONFERENCE

COMBINED PARTICIPATION BY:
DOD-ARMY-NAVY-AIR FORCE-NATO

30 NOVEMBER - 2 DECEMBER 1982
TUTORIALS: 29 NOVEMBER 1982

DAYTON CONVENTION CENTER
DAYTON, OHIO

TUTORIAL
MIL-STD-1589
JOVIAL (J-73) HIGH ORDER LANGUAGE

Approved for Public Release  Distribution Unlimited
NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely related Government procurement operation, the United States Government thereby incurs no responsibility nor any obligation whatsoever, and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture use, or sell any patented invention that may in any way be related thereto.

This report has been reviewed by the Office of Public Affairs (ASD/PA) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

Jeffery L. Pesler
JEFFERY L. PESLER
Vice Chairman
2nd AFSC Standardization Conference

E. C. Gangl
ERWIN C. GANGL
Chief, Avionics Systems Division
Directorate of Avionics Engineering

FOR THE COMMANDER

Robert P. Lavoie, COL, USAF
Director of Avionics Engineering
Deputy for Engineering

"If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization please notify ASD/ENAS, W-PAFB, OH 45433 to help us maintain a current mailing list".

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document."
UNCLASSIFIED

<table>
<thead>
<tr>
<th>REPORT DOCUMENTATION PAGE</th>
<th>READ INSTRUCTIONS BEFORE COMPLETING FORM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. REPORT NUMBER</td>
<td>2. GOVT ACCESSION NO</td>
</tr>
<tr>
<td>ASD(ENA)-TR-82-5031, VOLUME IV</td>
<td>A01142780</td>
</tr>
<tr>
<td>4. TITLE (and Subtitle)</td>
<td>5. TYPE OF REPORT &amp; PERIOD COVERED</td>
</tr>
<tr>
<td>Proceedings Papers of the Second AFSC Avionics Standardization Conference</td>
<td>Final Report</td>
</tr>
<tr>
<td></td>
<td>29 November - 2 December 1982</td>
</tr>
<tr>
<td>7. AUTHOR(S)</td>
<td>8. CONTRACT OR GRANT NUMBER(S)</td>
</tr>
<tr>
<td>Editor: Cynthia A. Porubcansky</td>
<td></td>
</tr>
<tr>
<td>9. PERFORMING ORGANIZATION NAME AND ADDRESS</td>
<td>10. PROGRAM ELEMENT, PROJECT, TASK AREA &amp; WORK UNIT NUMBERS</td>
</tr>
<tr>
<td>HQ ASD/ENAS</td>
<td></td>
</tr>
<tr>
<td>Wright-Patterson AFB OH 45433</td>
<td></td>
</tr>
<tr>
<td>11. CONTROLLING OFFICE NAME AND ADDRESS</td>
<td>12. REPORT DATE</td>
</tr>
<tr>
<td>HQ ASD/ENA</td>
<td>November 1982</td>
</tr>
<tr>
<td>Wright-Patterson AFB OH 45433</td>
<td></td>
</tr>
<tr>
<td>13. NUMBER OF PAGES</td>
<td>14. MONITORING AGENCY NAME &amp; ADDRESS (if different from Controlling Office)</td>
</tr>
<tr>
<td>Same as Above</td>
<td></td>
</tr>
</tbody>
</table>

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

N/A

18. SUPPLEMENTARY NOTES

N/A

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)


20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This is a collection of UNCLASSIFIED papers to be distributed to the attendees of the Second AFSC Avionics Standardization Conference at the Convention Center, Dayton, Ohio. The scope of the Conference includes the complete range of DoD approved embedded computer hardware/software and related interface standards as well as standard subsystems used within the Tri-Service community and NATO. The theme of the conference is "Rational Standardization". Lessons learned as well as the pros and cons of standardization are highlighted.
PROCEEDINGS OF THE

2nd AFSC
STANDARDIZATION CONFERENCE

30 NOVEMBER - 2 DECEMBER 1982

DAYTON CONVENTION CENTER
DAYTON, OHIO

Sponsored by: Air Force Systems Command
Hosted by: Aeronautical Systems Division
FOREWORD

THE UNITED STATES AIR FORCE HAS COMMITTED ITSELF TO "STANDARDIZATION." THE THEME OF THIS YEAR’S CONFERENCE IS "RATIONAL STANDARDIZATION," AND WE HAVE EXPANDED THE SCOPE TO INCLUDE US ARMY, US NAVY AND NATO PERSPECTIVES ON ONGOING DOD INITIATIVES IN THIS IMPORTANT AREA.

WHY DOES THE AIR FORCE SYSTEMS COMMAND SPONSOR THESE CONFERENCES? BECAUSE WE BELIEVE THAT THE COMMUNICATIONS GENERATED BY THESE GET-TOGETHERS IMPROVE THE ACCEPTANCE OF OUR NEW STANDARDS AND FOSTERS EARLIER, SUCCESSFUL IMPLEMENTATION IN NUMEROUS APPLICATIONS. WE WANT ALL PARTIES AFFECTED BY THESE STANDARDS TO KNOW JUST WHAT IS AVAILABLE TO SUPPORT THEM: THE HARDWARE; THE COMPLIANCE TESTING; THE TOOLS NECESSARY TO FACILITATE DESIGN, ETC. WE ALSO BELIEVE THAT FEEDBACK FROM PEOPLE WHO HAVE USED THEM IS ESSENTIAL TO OUR CONTINUED EFFORTS TO IMPROVE OUR STANDARDIZATION PROCESS. WE HOPE TO LEARN FROM OUR SUCCESSES AND OUR FAILURES; BUT FIRST, WE MUST KNOW WHAT THESE ARE AND WE COUNT ON YOU TO TELL US.

AS WE DID IN 1980, WE ARE FOCUSING OUR PRESENTATIONS ON GOVERNMENT AND INDUSTRY EXECUTIVES, MANAGERS, AND ENGINEERS AND OUR GOAL IS TO EDUCATE RATHER THAN PRESENT DETAILED TECHNICAL MATERIAL. WE ARE STRIVING TO PRESENT, IN A SINGLE FORUM, THE TOTAL AFSC STANDARDIZATION PICTURE FROM POLICY TO IMPLEMENTATION. WE HOPE THIS INSIGHT WILL ENABLE ALL OF YOU TO BETTER UNDERSTAND THE "WHY'S AND WHEREFORE'S" OF OUR CURRENT EMPHASIS ON THIS SUBJECT.

MANY THANKS TO A DEDICATED TEAM FROM THE DIRECTORATE OF AVIONICS ENGINEERING FOR ORGANIZING THIS CONFERENCE; FROM THE OUTSTANDING TECHNICAL PROGRAM TO THE UGLAMOROUS DETAILS NEEDED TO MAKE YOUR VISIT TO DAYTON, OHIO A PLEASANT ONE. THANKS ALSO TO ALL THE MODERATORS, SPEAKERS AND EXHIBITORS WHO RESPONDED IN SUCH A TIMELY MANNER TO ALL OF OUR PLEAS FOR ASSISTANCE.

ROBERT P. LAVOIE, COL, USAF
DIRECTOR OF AVIONICS ENGINEERING
DEPUTY FOR ENGINEERING
Second AFSC Standardization Conference

ASD/OC

1. Since the highly successful standardization conference hosted by ASD in 1980, significant technological advancements have occurred. Integration of the standards into weapon systems has become a reality. As a result, we have many "lessons learned" and cost/benefit analyses that should be shared within the tri-service community. Also, this would be a good opportunity to update current and potential "users." Therefore, I endorse the organization of the Second AFSC Standardization Conference.

2. This conference should cover the current accepted standards, results of recent congressional actions, and standards planned for the future. We should provide the latest information on policy, system applications, and lessons learned. The agenda should accommodate both government and industry inputs that criticize as well as support our efforts. Experts from the tri-service arena should be invited to present papers on the various topics. Our AFSC project officer, Maj David Hammond, HQ AFSC/ALR, AUTOVON 858-5731, is prepared to assist.

ROBERT M. BOND, Lt Gen, USAF
Vice Commander
MIL-STD-1589

JOVIAL (J-73) HIGH ORDER LANGUAGE

Instructor: Judy Bamberger
TRW/DSSG

ABSTRACT

An Introduction to the JOVIAL (J73) Programming Language presents an overview of the J73 language. Features common to many modern HOLs, such as strong typing, structured flow of control, modular program construction, are emphasized. The organization flows logically; first a brief preview of a complete program is presented, followed by a discussion of the building blocks of the language (declarations, executable statements, subroutines), concluding with a more thorough look at complete programs, and how the modularity constructs provided in J73 can be exploited to enhance the development of large software systems. Some of the more special-purpose features of the language are then briefly illustrated (e.g., built-in functions, specified tables). This introduction to J73 provides a logical view of the flavor and power of the J73 language for managers and programmers alike.

BIOGRAPHY

Judy Bamberger was born in Milwaukee Wisconsin on 26 September 1952. She received the B.S. degree in mathematics, French, and education from the University of Wisconsin-Milwaukee in 1974, and the M.Ed. degree in Junior High mathematics from the University of Northern Colorado (Greeley) in 1979.

From 1976 to 1979, she was a teacher in the Colorado school system. Then, from mid-1979 through early 1981, she joined SofTech Inc. in Waltham MA. There, she was responsible for all user documentation for the JOVIAL (J73) compilers. In addition, she developed a JOVIAL (J73) course, which she presented to several military and industrial organizations, both in this country and abroad. She designed and coordinated the production of the video course based on the original course. Since early 1981, she has been employed by TRW in Redondo Beach CA, where she was developing benchmark programs for JOVIAL compilers. She is currently part of the team developing a prototype of an advanced Ada Programming Support Environment (APSE) for the Navy.

Ms. Bamberger is an active member of the JOVIAL-Ada Users Group, where she is currently chairing the Education Committee.
AN INTRODUCTION TO THE
JOVIAL (J73)
PROGRAMMING LANGUAGE

Judy Bamberger
TRW
Redondo Beach CA
213-604-6251

Presented at The 2nd/AFSC Standardization Conference

29 November 1982
DISCLAIMER

This presentation DOES NOT:

- describe the syntax of JOVIAL (J73)
- discuss the more obscure points in the language
- illustrate the differences between different versions of J73 or other languages in detail

This presentation DOES:

- give an overview of the power and capabilities of the J73 language

Questions are welcome at any point!
WHAT IS HOL?

MACHINE LANGUAGE

actual binary instructions

01101110
01010000

ASSEMBLY LANGUAGE

more mnemonic instructions;
translated to machine instructions
by an assembler

L 12,FIRST
AX 12,0,6
STA ANSWER

HIGH ORDER LANGUAGE (HOL)

more English-like instructions;
translated to many assembly
language instructions by a
compiler

ANSWER = FIRST + OTHER;
WHY USE HIGH ORDER LANGUAGES (HOLs)?

- PROGRAMS ARE EASIER TO READ
  - more easily debugged
  - more easily maintained
  - don’t need to know the intricate details of the language in order to understand the code

- PROGRAMS ARE EASIER TO WRITE
  - HOL is learned more quickly
  - HOL coding is less error prone
  - HOL is coded more quickly

THUS HOL PROGRAMS HAVE LOWER LIFE-CYCLE COST
JOVIAL (J73)

1589A

1589B

1589C?
JOVIAL (J73) CAPABILITIES

- BLOCK-STRUCTURED PROGRAMS
  - subroutines - procedures and functions
  - modules - separate compilation units

- STRUCTURED CONTROL-FLOW STATEMENTS
  - loops
  - if
  - case

- STRONG TYPE CHECKING
  - restrictions on data conversions
    - user-definable types

- LOW-LEVEL OPERATIONS AND STORAGE DEFINITIONS
  - machine-specific subroutines
  - bit and byte manipulations
  - bit-level data description

- MACHINE PARAMETERS FOR PORTABILITY
JOVIAL (J73) CAPABILITIES

- MACROS (DEFINE CAPABILITY)
- NAME SCOPES
- COMPILER DIRECTIVES
  - listing formatting
  - optimization
  - module communication
- FREE FORMAT
  - indentation for legibility
  - single statements can continue over several lines
  - redundant blanks ignored
  - code can be easily commented
- NO BUILT-IN I/O
  - !! INKAGE
  - !TRACE
SAMPLE PROGRAM 1

main-program-module

compiler

assembler

linker

execute
SAMPLE PROGRAM 1

START
PROGRAM COUNTER;
BEGIN "MAIN-PROGRAM-MODULE"
"DECLARATIONS"
"EXECUTION"
"SUBROUTINES"
END "MAIN-PROGRAM-MODULE"
TERM
SAMPLE PROGRAM 1

-----------------------------

START

PROGRAM COUNTER;

BEGIN "MAIN-PROGRAM-MODULE"

"DECLARATIONS"

ITEM ONE S = 1;
ITEM TWO S = 2;
ITEM TOTAL S;

"EXECUTION"

TOTAL = ONE + TWO;

END "MAIN-PROGRAM-MODULE"

TERM
SAMPLE PROGRAM 1

START

PROGRAM COUNTER;

BEGIN   "MAIN-PROGRAM-MODULE"

"DECLARATIONS"

ITEM ONE S = 1;
ITEM TWO S = 2;
ITEM TOTAL S;

"EXECUTION"

COMPUTE (ONE, TWO : TOTAL);

"SUBROUTINES"

PROC COMPUTE (FIRST, SECOND : SUM);
BEGIN   "SUBROUTINE"
ITEM FIRST S;
ITEM SECOND S;
ITEM SUM S;

SUM = FIRST + SECOND;

END         "SUBROUTINE"
END         "MAIN-PROGRAM-MODULE"

TERM
SAMPLE PROGRAM 2

compool-module

START
COMPOOL DECLS;
:
TERM

procedure-module

START!
COMPOOL ('DECLS');
DEF PROC COMPUTE ..
:
TERM

main-program-module

START
!COMPOOL ('DECLS');
PROGRAM COUNTER;
:
TERM
SAMPLE PROGRAM 2

```
START
COMPOOL DECLS;
  DEF ITEM ONE S = 1;
  DEF ITEM TWO S = 2;
  DEF ITEM TOTAL;
  REF PROC COMPUTE
    (FIRST, SECOND :
      SUM);
  BEGIN
    ITEM FIRST S;
    ITEM SECOND S;
    ITEM SUM S;
  END
TERM
```

```
START
!COMPOOL ('DECLS');
  DEF PROC COMPUTE
    (FIRST, SECOND :
      SUM);
  BEGIN
    ITEM FIRST S;
    ITEM SECOND S;
    ITEM SUM S;
    SUM = FIRST + SECOND;
  END
TERM
```

```
START
!COMPOOL ('DECLS');
PROGRAM COUNTER;
BEGIN
  COMPUTE (ONE, TWO :
    THREE);
END
TERM
```

These modules are used to demonstrate the use of pool declarations and procedures in a program.
**J73 DATA OBJECTS**

<table>
<thead>
<tr>
<th>ITEM</th>
<th>simplest data object</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE</td>
<td>array, record, or array of records of ITEMS</td>
</tr>
<tr>
<td>BLOCK</td>
<td>group of ITEMs, TABLEs, and/or other BLOCKs</td>
</tr>
</tbody>
</table>

- declares the name and the attributes of a data object
- all data must be declared before it is used
- non-executable
- only one data object per declaration
VARIABLES AND CONSTANTS

VARIABLES
has storage allocated for it
possibly preset - if not, initial value
is undefined
referenced and set

CONSTANTS
possibly appears as immediate value in
assembly code and not allocated
preset to its constant value
referenced only
# DATA TYPES

<table>
<thead>
<tr>
<th>U</th>
<th>unsigned integer</th>
<th>0  --&gt; largest positive whole number</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>signed integer</td>
<td>smallest negative --&gt; 0 --&gt; largest positive whole number</td>
</tr>
<tr>
<td>F</td>
<td>floating point</td>
<td>fractional representation</td>
</tr>
<tr>
<td>A</td>
<td>fixed point</td>
<td>fractional representation – may use integer arithmetic</td>
</tr>
<tr>
<td>B</td>
<td>bit</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>character</td>
<td></td>
</tr>
<tr>
<td>STATUS</td>
<td>status (enumeration)</td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>pointer (access)</td>
<td></td>
</tr>
</tbody>
</table>
ITEM-DECLARATIONS

ITEM COUNTER U;
ITEM VARIANCE S 3;
ITEM VELOCITY F 23;
ITEM TARGET A 10, 3;
ITEM SWITCHES B 8;
ITEM LETTER C;
ITEM LAMP STATUS ( V(RED), V(YELLOW), V(GREEN) );

Item-declarations associate a name with a description of the type of the item.
ITEM-DECLARATIONS

ITEM

name

type of item

kind of data object

delimits values, operations, uses of the item

must be declared before used; associated with type; two or more characters

ITEM VELOCITY F 23;
ITEM VELOCITY F 23;
ITEM VELOCITY F 23;
ITEM-DECLARATIONS

ITEM COUNTER U = 6;
ITEM VARIANCE S 3 = -7 + 5;
ITEM VELOCITY F 23 = 2.0 + 3E2 - 9.7E-5;
CONSTANT ITEM TARGET A 10, 3 = 528.625;
ITEM SWITCHES B 10 = 1B'000111000';
CONSTANT ITEM LETTER C = 'Z';
ITEM LAMP STATUS ( V(RED), V(YELLOW), V(GREEN) ) = V(RED);

Items may be PRESET (given an initial value); they may be declared to be CONSTANTS, in which case they must be PRESET.
TYPE-DECLARATIONS

- declares a user-defined type-name
- no storage is allocated in a type-declaration

Advantages of type-names are:

more mnemonic
more structured
easier to change
ITEM TYPE-DECLARATIONS

TYPE SINGLE FLOAT F 23;
ITEM SPEED1 F 23;
ITEM SPEED2 SINGLE FLOAT;

TYPE COUNTTYPE U;
ITEM FLOOR COUNTTYPE
ITEM FLOOR COUNTTYPE = 3;
CONSTANT ITEM TEST\$LOOP COUNTTYPE = 7;

{ same type }
COMMENTS ON "TYPE"

JOVIAL (J73) is a "strongly typed" language.

The type of an item is used by the compiler throughout compilation to determine:

- legal values
- legal operations
- legal assignments

Correctly-declared data at the beginning avoids many problems later on; this is a "programmer beware" area of J73.

As we shall now see ...
ASSIGNMENT-STATEMENT

assigns a SOURCE (value, formula)
to a TARGET (variable)

TARGET = SOURCE;
TARGET1, TARGET2, ..., TARGET3 = SOURCE;

TYPE SINGLE'FLOAT F 23;

CONSTANT ITEM ZERO SINGLE'FLOAT = 0.0;
ITEM SPEED1 SINGLE'FLOAT;
ITEM SPEED2 SINGLE'FLOAT;
ITEM SPEED3 SINGLE'FLOAT;

SPEED1 = ZERO;
SPEED2, SPEED3 = ZERO;
ASSIGNMENT-STATEMENT

The type of the SOURCE must be EQUIVALENT or IMPLICITLY CONVERTIBLE to the type of the TARGET.

<table>
<thead>
<tr>
<th>EQUIVALENT types:</th>
<th>IMPLICITLY CONVERTIBLE types:</th>
</tr>
</thead>
<tbody>
<tr>
<td>U 3 = U 3</td>
<td>U 10 = U 8</td>
</tr>
<tr>
<td>S 24 = S 24</td>
<td>U 2 = U 15</td>
</tr>
<tr>
<td></td>
<td>S 12 = U 10</td>
</tr>
<tr>
<td></td>
<td>U 7 = S 20</td>
</tr>
<tr>
<td></td>
<td>F 23 = F 3</td>
</tr>
<tr>
<td>F 7 = F 7</td>
<td>A 6, 7 = A 3, 2</td>
</tr>
<tr>
<td>A 10, 2 = A 10, 2</td>
<td>B 10 = B 2</td>
</tr>
<tr>
<td>B 8 = B 8</td>
<td>B 8 = B 64</td>
</tr>
<tr>
<td>C 3 = C 3</td>
<td>C 2 = C 9</td>
</tr>
<tr>
<td></td>
<td>C 7 = C 3</td>
</tr>
</tbody>
</table>
TYPE EQUIVALENCE, IMPLICIT AND EXPLICIT CONVERSION

EQUIVALENT

Types are defined to be "identical"
No action by programmer or compiler is required

IMPLICITLY CONVERTIBLE

Types are defined to be "close"
No action by programmer is required; the compiler may automatically do something to make the types equivalent

EXPLICITLY CONVERTIBLE

Types are defined to be "different"
Programmer must code an EXPLICIT CONVERSION; the compiler acts on that information
ASSIGNMENT-STATEMENT

The following are ILLEGAL assignments:

floating point = integer
    bit = integer
short float = long float
    character = bit
    integer = bit

... but there may be times when a programmer needs to access the value of a data object of one type as another type.
ITEM WEIGHT F 23 = 62.732;
ITEM INT'WEIGHT S;
TYPE S'WORD S;

INT'WEIGHT = WEIGHT; <---------- ILLEGAL

INT'WEIGHT = (* S 15 *) (WEIGHT);
INT'WEIGHT = (* S * ) (WEIGHT);
INT'WEIGHT = S (WEIGHT);
INT'WEIGHT = S'WORD (WEIGHT);
INT'WEIGHT = (* S'WORD *) (WEIGHT);
INT'WEIGHT = (* S, T *) (WEIGHT);

INT'WEIGHT = (* S, R *) (WEIGHT);
CONVERTIBLE DATA TYPE TABLE

<table>
<thead>
<tr>
<th>SOURCE (from) type</th>
<th>U short</th>
<th>U long</th>
<th>S short</th>
<th>S long</th>
<th>F short</th>
<th>F long</th>
<th>A short</th>
<th>A long</th>
<th>B short</th>
<th>B long</th>
<th>C short</th>
<th>C long</th>
</tr>
</thead>
<tbody>
<tr>
<td>U short</td>
<td>E I I I X X X</td>
<td>X X X</td>
<td>E I I I X X X</td>
<td>X X X</td>
<td>E I I I X X X</td>
<td>X X X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>U long</td>
<td>I E I I X X X</td>
<td>X X X</td>
<td>I E I I X X X</td>
<td>X X X</td>
<td>I E I I X X X</td>
<td>X X X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S short</td>
<td>I I E I X X X</td>
<td>X X X</td>
<td>I I E I X X X</td>
<td>X X X</td>
<td>I I E I X X X</td>
<td>X X X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S long</td>
<td>I I I E X X X</td>
<td>X X X</td>
<td>I I I E X X X</td>
<td>X X X</td>
<td>I I I E X X X</td>
<td>X X X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F short</td>
<td>X X X X E I X X</td>
<td>X X X</td>
<td>X X X X I E X X</td>
<td>X X X</td>
<td>X X X X I E X X</td>
<td>X X X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F long</td>
<td>X X X X I E X X</td>
<td>X X X</td>
<td>X X X X I E X X</td>
<td>X X X</td>
<td>X X X X I E X X</td>
<td>X X X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A short</td>
<td>X X X X E I X X</td>
<td>X X X</td>
<td>X X X X I E X X</td>
<td>X X X</td>
<td>X X X X I E X X</td>
<td>X X X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A long</td>
<td>X X X X I E X X</td>
<td>X X X</td>
<td>X X X X I E X X</td>
<td>X X X</td>
<td>X X X X I E X X</td>
<td>X X X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B short</td>
<td>* * * * * * E I X X</td>
<td>X X X</td>
<td>* * * * * * I E X X</td>
<td>X X X</td>
<td>* * * * * * I E X X</td>
<td>X X X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B long</td>
<td>* * * * * * I E X X</td>
<td>X X X</td>
<td>* * * * * * I E X X</td>
<td>X X X</td>
<td>* * * * * * I E X X</td>
<td>X X X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C short</td>
<td></td>
<td></td>
<td></td>
<td>X X E I</td>
<td>X X X I E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C long</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

short and long are relative terms only
there are special rules for STATUS (and pointer) types

E = equivalent types
I = implicitly convertible types
X = explicitly convertible types
* = explicitly convertible types with restrictions
FORMULAE

U, S
(integer)

+  addition
-  subtraction
*  multiplication
/  integer division
** integer exponentiation
MOD  modulus – integer remainder of integer division

Integer formulae take any integer operands and return a result of type integer. (There are special restrictions on integer exponentiation in 1589A implementations.)
FORMULAE

ITEM ONE S = 1;
ITEM TWO S 6 = 2;
ITEM THREE U 15 = 3;
ITEM RESULT S 31;

RESULT = ONE + TWO + (-5);
RESULT = (THREE - THREE) * TWO;
RESULT = THREE / TWO;
RESULT = TWO ** 7;
RESULT = (16 MOD (THREE + TWO)) * 2;
FORMULAE

F
(floating point)

+ addition
- subtraction
* multiplication
/ division
** all other exponentiations

Floating point formulae take any floating point operands and return a result of type floating point. In addition, the exponent in exponentiation may be an integer. (1589A implementations use floating point exponentiation for some cases where both operands are integers.)
FORMULAE

ITEM ONE F 23 = 1.0;
ITEM TWO F 7 = .2E1;
ITEM THREE F 15 = 3E0;
ITEM RESULT F 23;

RESULT = ONE + TWO + (-5.000E0);
RESULT = (THREE - THREE) * TWO;
RESULT = THREE / TWO;
RESULT = TWO ** 7;
RESULT = (-.47E10 / (TWO + THREE)) - ONE;
FORMULAE

A
(fixed point)

+  addition
-  subtraction
*  multiplication
/  division

Fixed point formulae take fixed point or integer operands, depending on the operator; there are other restrictions on operands for addition and subtraction. The type of the result of a fixed point formula is fixed point.
FORMULAE

ITEM ONE'FOURTH A 3, 2 = .25;
ITEM ONE'EIGHTH A 3, 5 = 125E-3;
ITEM TWO S = 2;
ITEM THREE S = 3;
ITEM FOUR'AND'ONE'HALF A 7, 4 = 4.5;
ITEM RESULT1 A 3, 8;
ITEM RESULT2 A 10, 6;

RESULT1 = ONE'FOURTH + ONE'EIGHTH;
RESULT1 = 3.5 - ONE'EIGHTH;
RESULT1 = THREE * ONE'FOURTH;
RESULT2 = ONE'FOURTH * FOUR'AND'ONE'HALF;
RESULT1 = (ONE'EIGHTH / TWO) + 3125E-3;
RESULT2 = (* A 10, 6 *) (ONE'FOURTH / ONE'EIGHTH);
FORMULAE

B (bit)       AND    logical "and"
            OR      logical "or"
            XOR     logical "exclusive or"
            EQV     logical "equivalence"
            NOT     logical "not"

Bit formulae take any bit operands and return a result of type bit.
ITEM B5 = B1'B10101';
ITEM B10 B 10 = B1'B11100000';
ITEM B10'TOO B 10 = B1'B10101010';
ITEM BOOLEAN B 1 = TRUE;
ITEM BOOLEANF B 1 = FALSE;
ITEM RESULT1 B 1;
RESULT1 = BOOLEAN AND BOOLEANF;
" 1B'0';
RESULT2 = B10 EQV B10'TOO;
RESULT1 = BOOLEAN OR 1B';
RESULT2 = (B10 XOR B10'TOO) AND B5;
RESULT1 = NOT BOOLEAN;
RESULT1 = BOOLEAN AND (NOT BOOLEANF);
" 1B'0';
OPERATOR PRECEDENCE

** 5
* / MOD 4
+ - 3
NOT AND OR XOR EQV 1

- operators at higher precedence are evaluated first
- no precedence among logical operators
- formulae may be parenthesized
ITEM ANSWER1 S 15;
ITEM ANSWER2 S 15;

ANSWER1 = 67;
ANSWER2 = (-4, -12);
ANSWER1 = ANSWER1, ANSWER2;
TARGET(s) ←------- SOURCE
RELATIONAL EXPRESSION

operators:  >
            >=
            <
            <=
            <>

operands must be equivalent or implicitly convertible
returns Boolean TRUE or FALSE

(#2 on precedence of operators chart)
RELATIONAL EXPRESSION

ITEM UU U 15 = 6;
ITEM SS S = -37;
ITEM FF F 23 = 27.5E2;
ITEM AA A 12, 3 = 4.25;
ITEM BB B 2 = 1B’11’;
ITEM CC C = 'Z'
ITEM ST STATUS ( V(A), V(B), V(C), V(D) ) = V(C);

UU >= SS  --------> TRUE
SS = 10  --------> FALSE
FF <> 25.0  --------> TRUE
AA < 1E-3  --------> FALSE
BB = 1B'11'  --------> TRUE
CC < 'A'  --------> TRUE
ST <= V(D)  --------> TRUE
IF-STATEMENT

TYPE FLOAT'TYPE F 23;
ITEM SUM FLOAT'TYPE;
ITEM LIMIT FLOAT'TYPE = 400.0;
ITEM ANSWER FLOAT'TYPE;

IF SUM > LIMIT;
    SUM = SUM / 2.0;
ELSE
    SUM = SUM + 1.0;

ANSWER = SUM;

if SUM = 500.0,
    ANSWER = 250.0
if SUM = 300.0,
    ANSWER = 301.0
IF-STATEMENT

TYPE FLOAT'TYPE F 23;
ITEM SUM FLOAT'TYPE;
ITEM LIMIT FLOAT'TYPE = 400.0;
ITEM ANSWER FLOAT'TYPE;

IF SUM > LIMIT;
    SUM = SUM / 2.0;

ANSWER = SUM;

if SUM = 500.0,
    ANSWER = 250.0

if SUM = 300.0,
    ANSWER = 300.0
IF-STATEMENT

TYPE LETTER STATUS ( V(A), V(B), V(C) );
ITEM GRADE LETTER;
ITEM CO'OPERATIVE B 1;
ITEM REPORT'CARD C 2;

IF GRADE = V(A);
    IF CO'OPERATIVE = TRUE;
        REPORT'CARD = 'A+';
    ELSE "IF CO'OPERATIVE = FALSE"
        REPORT'CARD = 'A';
ELSE
    IF GRADE = V(B);
        IF CO'OPERATIVE = TRUE;
            REPORT'CARD = 'B+';
        ELSE "IF CO'OPERATIVE = FALSE"
            REPORT'CARD = 'B';
    ELSE
        IF GRADE = V(C);
            IF CO'OPERATIVE = TRUE;
                REPORT'CARD = 'C+';
            ELSE "IF CO'OPERATIVE = FALSE"
                REPORT'CARD = 'C';
IF-STATEMENT

TYPE LETTER STATUS ( V(A), V(B), V(C) );
ITEM GRADE LETTER;
ITEM CO'OPERATIVE B 1;
ITEM REPORT'CARD C 2;

IF ((GRADE = V(A)) AND (CO'OPERATIVE));
   REPORT'CARD = 'A+';
ELSE
   "IF CO'OPERATIVE = FALSE"
   REPORT'CARD = 'A';

IF ((GRADE = V(B)) AND (CO'OPERATIVE));
   REPORT'CARD = 'B+';
ELSE
   "IF CO'OPERATIVE = FALSE"
   REPORT'CARD = 'B';

IF ((GRADE = V(C)) AND (CO'OPERATIVE));
   REPORT'CARD = 'C+';
ELSE
   "IF CO'OPERATIVE = FALSE"
   REPORT'CARD = 'C';
COMPOUND-STATEMENTS

BEGIN

simple-statements

END

- groups more than one simple-statement to be treated as a single syntactic entity
IF-STATEMENT

TYPE FLOAT'TYPE F 23;
ITEM SUM FLOAT'TYPE;
ITEM LIMIT FLOAT'TYPE = 400.0;
ITEM ANSWER FLOAT'TYPE;
ITEM OVERFLOW B 1;

IF SUM > LIMIT;
   BEGIN
      SUM = SUM / 2.0;
      OVERFLOW = TRUE;
   END
ELSE
   BEGIN
      SUM = SUM + 1.0;
      OVERFLOW = FALSE;
   END

ANSWER = SUM;
GOTO-STATEMENT

TYPE FLOAT'TYPE F 23;
ITEM SUM FLOAT'TYPE;
ITEM LIMIT FLOAT'TYPE = 400.0;
ITEM ANSWER FLOAT'TYPE;
ITEM OVERFLOW B 1;

IF SUM > LIMIT;
  BEGIN
    SUM = SUM / 2.0;
    OVERFLOW = TRUE;
  END
ELSE
  BEGIN
    IF SUM = LIMIT;
      GOTO SET'ANSWER;
    SUM = SUM + 1.0;
    OVERFLOW = FALSE;
  END
SET'ANSWER:
  ANSWER = SUM;
CASE-STATEMENT

TYPE U'WORD U;
ITEM NUMBER U'WORD;
ITEM COUNT U'WORD;

CASE NUMBER;
BEGIN
  (DEFAULT): COUNT = 0;
  (1, 2): COUNT = COUNT + 1;
  (3 : 5): COUNT = COUNT + 2;
END
IF-STATEMENT

IF (NUMBER = 1) OR (NUMBER = 2);
COUNT = COUNT + 1;
ELSE
  IF (NUMBER >= 3) AND (NUMBER <= 5);
  COUNT = COUNT + 2;
ELSE
  COUNT = 0;

This if-statement corresponds to the preceding case-statement. The implementation of the case-statement handles all the if tests; the programmer does not need to code them.
CASE-STATEMENT

-----------------------------

case-selector is evaluated

case-selector is tested against each of the
case-indices

if a "match" is found, the appropriate case-option
is executed, and processing continues after the
case-statement

if a "match" is not found, the default-option is
executed, and processing continues after the
case-statement

if a "match" is not found and no default-option
is present, programmer beware!
CASE-SELECTOR AND CASE-INDICES

| type: | U | S | B | C | STATUS |

- type of case-selector must be EQUIVALENT or IMPLICITLY CONVERTIBLE to the type of the case-indices
CASE-INDICES

- single values
- enumerated values
- range of values (U, S, and some STATUS only)
- values known at compile-time
- distinct between case-options
CASE-STATEMENT

TYPE S'WORD S 15;
ITEM COUNTER S'WORD;
ITEM NUMBER S'WORD;
ITEM CATEGORY C;

COUNTER, NUMBER = 0;

CASE CATEGORY;
  BEGIN
  (DEFAULT): ;
  ('A', 'B'): BEGIN
                COUNTER = COUNTER + 1;
                NUMBER = NUMBER + 1;
                END FALLTHRU
  ('C'):     COUNTER = COUNTER + 3; FALLTHRU
  ('D', 'E', 'F'): BEGIN
                   COUNTER = COUNTER + 5;
                   NUMBER = NUMBER + 2;
                   END
    END
  "CASE"
## CASE-STATEMENT

<table>
<thead>
<tr>
<th>CASE</th>
<th>COUNTER</th>
<th>NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>if CATEGORY = 'G',</td>
<td>COUNTER = 0</td>
<td>NUMBER = 0</td>
</tr>
<tr>
<td>if CATEGORY = 'B',</td>
<td>COUNTER = 9</td>
<td>NUMBER = 3</td>
</tr>
<tr>
<td>if CATEGORY = 'C',</td>
<td>COUNTER = 8</td>
<td>NUMBER = 2</td>
</tr>
<tr>
<td>if CATEGORY = 'E',</td>
<td>COUNTER = 5</td>
<td>NUMBER = 2</td>
</tr>
</tbody>
</table>
LOOP-STATEMENTS

repeated execution of a controlled-statement

two kinds of loops:

while-loop
for-loop
WHILE-LOOP

TYPE S'WORD S 15;
ITEM COUNTER S'WORD = 1;
ITEM SUM S'WORD = 0;
CONSTANT ITEM LIMIT S'WORD = 10;

WHILE SUM < LIMIT;
BEGIN
  COUNTER = COUNTER + 1;
  SUM = SUM + COUNTER;
END
WHILE-LOOP

1) test while-phrase
2) if TRUE, execute controlled-statement
3) loop
4) if FALSE, exit loop

FALSE

SUM < LIMIT

TRUE

COUNTER = COUNTER + 1
SUM = SUM + COUNTER

continue
FOR-LOOP

TYPE U'WORD U 15;
ITEM COUNTER U'WORD;
ITEM TOTAL U'WORD = 0;

FOR COUNTER : 1 BY 2 WHILE COUNTER <= 10;
   TOTAL = TOTAL + 1;
FOR-LOOP

1) initialize loop-control
2) test while-phrase
3) if TRUE, execute controlled-statement
4) increment loop-control
5) loop
3') if FALSE, exit loop

COUNTER = 1

COUNTER <= 10

TOTAL = TOTAL + 1

COUNTER = COUNTER + 2

FALSE

TRUE

continue
FOR-LOOP

TYPE S'WORD S 15;
ITEM XX S'WORD;
ITEM YY S'WORD;
ITEM RESULT S'WORD;
ITEM FOUND B 1 = FALSE;
ITEM X'LOOP S'WORD;
ITEM Y'LOOP S'WORD;

"THIS NESTED LOOP FINDS THE FIRST SOLUTION TO THE
EQUATION 3X - 4Y = 0 FOR X AND Y IN THE RANGE 1 - 5"

FOR X'LOOP : 1 BY 1 WHILE (X'LOOP <= 5 AND FOUND = FALSE);

FOR Y'LOOP : 1 BY 1 WHILE (Y'LOOP <= 5 AND FOUND = FALSE);

BEGIN "INNER LOOP"
RESULT = (3 * X'LOOP) - (4 * Y'LOOP);
IF RESULT = 0;
BEGIN "FOUND A SOLUTION"
XX = X'LOOP;
YY = Y'LOOP;
FOUND = TRUE;
END "FOUND A SOLUTION"
END "INNER LOOP"
FOR-LOOP

TYPE SINGLE’FLOAT F 23;
ITEM NUMBER SINGLE’FLOAT;
ITEM SUM SINGLE’FLOAT = 0.0;

FOR NUMBER : 2.0 THEN NUMBER ** 2 WHILE NUMBER < 200.0;
SUM = SUM + NUMBER;
FOR-LOOP

1) initialize loop-control

2) test while-phrase

3) if TRUE, execute controlled-statement

4) reassign loop-control

5) loop

3'') if FALSE, exit loop

\[
\text{NUMBER = 2.0}
\]

\[
\text{NUMBER < 200.0}
\]

\[
\text{SUM = SUM + NUMBER}
\]

\[
\text{NUMBER = NUMBER ** 2}
\]

continue
FOR-LOOP

flow of control through any for-loop
- initialize loop-control
- evaluate condition in while-phrase
  - if TRUE
    - execute controlled-statement
    - increment loop-control (BY)
    - or -
    - reassign loop-control (THEN)
- loop to test while-phrase
  - if FALSE
    - exit loop
FOR-LOOP

by-clause           U, S, F, A only
                   the value is ADDED TO loop-control

then-clause         any type
                   the value is REASSIGNED TO loop-control

The type of initial-value must be EQUIVALENT or IMPLICITLY CONVERTIBLE to the type of loop-control; the type of the formula in the by-or-then-clause must be EQUIVALENT or IMPLICITLY CONVERTIBLE to the type of loop-control.
FOR-LOOP

-----> FOR INDEX : 10 BY -1;
       COUNT = COUNT + 1;

-----> FOR NUMBER : 15.0 WHILE NUMBER <= 40.0;
       SUM = SUM * 4.32;

The while-phrase and by-or-then-clause may be omitted in a loop; other means of altering loop-control and testing for loop termination should be used.
FOR-LOOP

- single-letter loop-control
  - implicitly declared by its use
  - type is type of initial-value
  - value is NOT known outside of loop
  - value can NOT be changed in controlled-statement

FOR I : 10 BY -1 WHILE I <> 0;
  BEGIN
    SUM = SUM + I;
    :
  END
EXIT-STATEMENT

controlled, premature exit from the currently-executing controlled-statement

FOR I : 10.5 BY 0.5 WHILE I < 100.0;
  BEGIN
  SUM = SUM + I;
  IF SUM > 225.5;
    EXIT;
  END
TABLES

record-like

TABLE TAB1;
    BEGIN
    ITEM NAME C 30;
    ITEM RANK C 5;
    ITEM SERIAL'NUMBER C 9;
    END

array-like

TABLE VECTOR (1 : 20);
    ITEM VECTOR'1 F 23;
array of records

TABLE HOUSES (1 : 9);
BEGIN
ITEM ROOM STATUS ( V(LIVING), V(KITCHEN),
                   V(BED1), V(BED2) );
ITEM LENGTH F 23;
ITEM WIDTH F 23;
ITEM HEIGHT F 23;
END
TABLE-DECLARATION

TABLE TAB1;
      BEGIN
      ITEM NAME C 30; 1 entry
      ITEM RANK C 5; 3 items / entry
      ITEM SERIAL'NUMBER C 9;
      END

TYPE S'WORD S 15;
TABLE TAB2;
      ITEM VALUE S'WORD;
      1 entry
      1 item / entry

      table-body may be compound (BEGIN-END) or simple
TABLE DECLARATION

TABLE TAB1;
BEGIN
ITEM NAME C 30 = 'MR. X';
ITEM RANK C 5;
ITEM SERIAL'NUMBER C 9 = '112233344';
END

TABLE TAB1 = 'MR. X', '112233344';
BEGIN
ITEM NAME C 30;
ITEM RANK C 5;
ITEM SERIAL'NUMBER C 9;
END

non-preset items have undefined initial values

not all items need to be preset
TABLE-DECLARATION

TABLE GRADES;
BEGIN
ITEM MATH C 1;
ITEM ENGLISH C 1;
ITEM HISTORY C 1;
ITEM SCIENCE C 1;
ITEM ART C 1;
ITEM MUSIC C 1;
END

a repetition count may be used in a table-preset

TABLE GRADES = 6 ('A');
    BEGIN
    :
    END

TABLE GRADES = 2 ('A', 'B');
    BEGIN
    :
    END

TABLE GRADES = 2 ('A', 2 ('B'));
    BEGIN
    :

TABLE-DECLARATION

=================================

tables may be declared to be constant

CONSTANT TABLE LOOK'UP;
BEGIN
ITEM NUMBER F 23 = 4.0;
ITEM SQUARE F 23 = 16.0;
ITEM CUBE F 23 = 64.0;
ITEM RECIPROCAL F 23 = 1.0 / 4.0;
ITEM ROOT F 23 = 4.0 ** (0.5);
END
DATA REFERENCES

TOTAL = NUMBER + ROOT;
FOR LENGTH : 1.0 BY SQUARE WHILE LENGTH < CUBE;
RESULT = (* S 15 *) (SQUARE);
MATH = 'C';
MUSIC = ART;

simple data references are used for items in record-like tables
TABLE TYPE-DECLARATIONS

sets up a "template" that can be used to declare
any number of tables with the same table-entry

TYPE LOCATION'TYPE TABLE;
BEGIN
ITEM LONGITUDE F 23;
ITEM LATITUDE F 23;
END

TABLE WORK LOCATION'TYPE;

TABLE MAP LOCATION'TYPE;

TABLE NORTH'POLE LOCATION'TYPE = 0.0, 0.0;
TABLE TYPE-DECLARATIONS

like-option may be used to describe table-entries
like another type, possibly with additional items

TYPE GRADE STATUS ( V(A), V(B), V(C), V(D), V(F) );

TYPE BASICS TABLE;
   BEGIN
   ITEM READING GRADE;
   ITEM WRITING GRADE;
   ITEM RITHMETIC GRADE;
   END

TYPE ADVANCED TABLE LIKE BASICS;
   BEGIN
   ITEM SCIENCE GRADE;
   ITEM HISTORY GRADE;
   END

TABLE KENNETH BASICS;

TABLE JOHN ADVANCED;
A QUICK NOTE ON TYPED TABLES

must use POINTERS to reference objects in typed tables

will not be covered in this presentation
### TABLE-DECLARATION

<table>
<thead>
<tr>
<th>Table</th>
<th>Entries</th>
<th>Items / Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE VECTOR (1 : 20)</td>
<td>20</td>
<td>1</td>
</tr>
<tr>
<td>ITEM VECTOR'I F 23;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TABLE MATRIX (1 : 4, 1 : 4);</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>ITEM MATRIX'I S 15;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In table VECTOR, there are 20 items VECTOR'I; a SUBSCRIPT must be used to reference a particular VECTOR'I. (Similar for table MATRIX and its 16 items MATRIX'I.)

In J73 - RIGHTMOST SUBSCRIPT VARIES FIRST.
SUBSCRIPTS

VECTOR'1 (3) = 32E7;

VECTOR'1 (17) = VECTOR'1 ((0 + 9 - 7) / 2);

RESULT = VECTOR'1 (2) * VECTOR'1 (20);

MATRIX'1 (1, 3) = -46;

MATRIX'1 (1, 1) = MATRIX'1 (4, 4) + MATRIX'1 (2, 3);

FOR I : 1 BY 1 WHILE I <= 4;
    FOR J : 1 BY 1 WHILE J <= 4;
        MATRIX (I, J) = 0;
tables may be preset ...

**TABLE MATRIX** (1 : 4, 1 : 4);
ITEM MATRIX's 15 = 4 (1, 0, 0, 0);

**TABLE MATRIX** (1 : 4, 1 : 4) = 4 (1, 0, 0, 0);
ITEM MATRIX's 15;

**TABLE MATRIX** (1 : 4, 1 : 4);
ITEM MATRIX's 15 = POS (1, 1): 1,
POS (2, 1): 1,
POS (3, 1): 1,
POS (4, 1): 1, 0, 0, 0,
POS (3, 2): 3 (0),
POS (2, 2): 3 (0),
POS (1, 2): 3 (0);

<table>
<thead>
<tr>
<th></th>
<th>1.1</th>
<th>1.2</th>
<th>1.3</th>
<th>1.4</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
MICROCOPY RESOLUTION TEST CHART

NATIONAL BUREAU OF STANDARDS 1974

1.0
1.1
1.25
1.4
1.6

1.8
2.0
2.2
2.5
TABLE-DECLARATION

------------------------------

TABLE HOUSES (1 : 9);
BEGIN
ITEM ROOM STATUS ( V(LIVING), V(KITCHEN),
                  V(BED1), V(BED2) );
ITEM LENGTH F 23;
ITEM WIDTH F 23;
ITEM HEIGHT F 23;
END

9 entries
4 items / entry

TABLE CLASS'RECORD (1 : 5, 1 : 4);
BEGIN
ITEM STUDENT'NAME C 20;
ITEM AGE S;
ITEM SEX STATUS ( V(MALE), V(FEMALE) );
END

20 entries
3 items / entry
SUBSCRIPTS

LENGTH (9) = 14.5;
WIDTH (4) = HEIGHT (1);
AREA = LENGTH (3) * WIDTH (3);

FOR I : 1 BY 1 WHILE I <= 5;
   FOR J : 1 BY 1 WHILE J <= 5;
      SUM = SUM + AGE (I, J);

SEX (5, 4) = V(MALE);
the first two entries are preset

TABLE HOUSES (1: 9);
BEGIN
ITEM ROOM STATUS (V(LIVING), V(KITCHEN),
    V(BED1), V(BED2)) =
    V(LIVING), V(KITCHEN);
ITEM LENGTH F 23 = 12.5, 10.5;
ITEM WIDTH F 23 = 2 (8.0);
ITEM HEIGHT F 23 = 2 (7.0);
END

TABLE HOUSES (1: 9) = V(LIVING), 12.5, 8.0, 7.0,
    V(KITCHEN), 10.5, 8.0, 7.0;
BEGIN
: 
END

TABLE HOUSES (1: 9) = POS (2): V(KITCHEN), 10.5,
    POS (1): V(LIVING), 12.5,
    POS (2): , , 8.0, 7.0,
    POS (1): , , 8.0, 7.0;
BEGIN
: 
END
TABLE TYPE-DECLARATION

Table entry only (record)

- entire table

TYPE AREA TABLE;
BEGIN
ITEM LENGTH S;
ITEM WIDTH S;
END

TABLE ONE‘FLOOR AREA; table is typed

TABLE NINE‘FLOORS (1 : 9) AREA; entries are typed
TABLE TYPE-DECLARATION

TYPE AREA TABLE;
BEGIN
ITEM LENGTH S;
ITEM WIDTH S;
END

TYPE FLOORS TABLE (1 : 9) AREA;

TABLE ONE’FLOOR AREA; table is typed (AREA)
TABLE NINE’FLOORS FLOORS; table is typed (FLOORS)
entries are typed (AREA)
TABLE ALSO’NINE’FLOORS (1 : 9) AREA;
entries are typed (AREA)
TABLE TYPE-DECLARATION

like-option may be used to describe table-entries
like another type, possibly with additional items

TYPE AREA TABLE;
  BEGIN
  ITEM LENGTH S;
  ITEM WIDTH S;
  END

TYPE MEASURES TABLE (1 : 4) LIKE AREA;
  ITEM HEIGHT S;

TYPE APARTMENTS TABLE LIKE MEASURES;
  BEGIN
  ITEM BUILDING'NO C 1;
  ITEM ADDRESS C 50;
  END
TABLE TYPE-DECLARATION

TABLE ONE'ROOM AREA;
1 entry
2 items / entry

TABLE MANY'ROOMS (1 : 3, 1 : 4) AREA;
12 entries
2 items / entry

TABLE FIRST'FLOOR MEASURES;
4 entries
3 items / entry

TABLE INVESTMENTS APARTMENTS;
4 entries
5 items / entry

- only one dimension-list per table, whether it comes from table-declaration, table type-declaration, or like-option
SAMPLE PROGRAM 1

START
PROGRAM FIND'FACTORS;
BEGIN
  "PROGRAM"

  "DECLARATIONS"

  CONSTANT ITEM NUMBER'TO'FACTOR U = 24;
  TABLE FACTOR'TAB (1 : NUMBER'TO'FACTOR);
    ITEM FACTOR B 1 = NUMBER'TO'FACTOR (FALSE);
    ITEM FACTOR'LIMIT U;

  "EXECUTION"

  FACTOR'LIMIT = (* S *) ((* F *) (NUMBER'TO'FACTOR) ** (0.5));
  FOR I : 1 BY 1 WHILE I <= FACTOR'LIMIT;
    IF (NUMBER'TO'FACTOR MOD I = 0);
      BEGIN
        "FOUND A FACTOR"
        FACTOR (I) = TRUE;
        FACTOR (NUMBER'TO'FACTOR / I) = TRUE;
      END
    END
  "FOUND A FACTOR"
END
"PROGRAM"
TERM
SAMPLE PROGRAM 2

START
PROGRAM MATRIX'ADDITION;
BEGIN
"PROGRAM"

"DECLARATIONS"

TABLE MATRIX1 (1 : 3, 1 : 3) = 9 (5.0);
ITEM MATRIX1'I F;
TABLE MATRIX2 (1 : 3, 1 : 3) = 3 (1.0, 2.0, 3.0);
ITEM MATRIX2'I F;
TABLE MATRIX'ANSWER (1 : 3, 1 : 3);
ITEM MATRIX'ANSWER'I F;

"EXECUTION"

FOR 1 : 1 BY 1 WHILE I <= 3;
FOR J : 1 BY 1 WHILE J <= 3;
 MATRIX'ANSWER'I (I, J) =
    MATRIX1'I (I, J) + MATRIX2'I (I, J);
END
"PROGRAM"
SAMPLE PROGRAM 3

START
PROGRAM AREAS;
BEGIN

"PROGRAM"

"DECLARATIONS"

TYPE FLOAT'TYPE F;
TYPE SHAPE'TYPE STATUS ( V(SQUARE), V(RECTANGLE),

V/TRIANGLE), V(OTHER) );

TABLE RESULTS (1 : 4);
BEGIN
ITEM SHAPE SHAPE'TYPE = V(OTHER),

V(RECTANGLE),
V(SQUARE),

V/TRIANGLE);

ITEM AREA FLOAT'TYPE;
ITEM SIDE1 FLOAT'TYPE = 4.0, 9.5, 8.0, 6.3;
ITEM SIDE2 FLOAT'TYPE = 2.0, 4.0;
END
FOR Z : 1 BY 1 WHILE Z <= 4;
CASE SHAPE (I);
BEGIN

"CASE"
( DEFAULT):
(V(TRIANGLE)):
AREA (I) = 0.5 * SIDE1 (I) *
SIDE2 (I);
(V(SQUARE)):
AREA (I) = SIDE1 (I) * SIDE1 (I);
(V(RECTANGLE)):
AREA (I) = SIDE1 (I) * SIDE2 (I);
END
"CASE"
"PROGRAM"

TERM

99
PROGRAM ORGANIZATION

START
PROGRAM name;
BEGIN
"DECLARATIONS"
"EXECUTABLE STATEMENTS"
"SUBROUTINES"
END
TERM
PROGRAM ORGANIZATION

START
PROGRAM name;
BEGIN
"PROGRAM"

"DECLARATIONS"
"EXECUTABLE STATEMENTS"
"SUBROUTINES"

PROC name;
BEGIN
"SUBROUTINE"

"DECLARATIONS"
"EXECUTABLE STATEMENTS"
"SUBROUTINES"

END
"SUBROUTINE"

END
"PROGRAM"

TERM

- a subroutine is like a small program
SUBROUTINES

- provides modularity
- improves program organization
- may perform similar sequence of action at different places in program
- may be "parameterized" to perform same computation on different sets of data
- subroutine is a generic term for
  - procedure
  - function (returns a value)
SUBROUTINE-DEFINITION

procedure - PROC EXAMPLE'PROC;
BEGIN
"DECLARATIONS"
"EXECUTION"
END

function - PROC EXAMPLE'FUNC U 15;
BEGIN
"DECLARATIONS"
"EXECUTION"
END

function-definition has a type associated with the function-name
SUBROUTINE INVOCATION

procedure -

       ...

       EXAMPLE'PROC;

function - ITEM ANSWER U;

       ...

       ANSWER = EXAMPLE'FUNC;

- a subroutine is not executed until it is invoked (called)
SUBROUTINE TERMINATION

正常
- 执行子例行程序的最后一个语句
- 执行返回语句

...
...
PROC COUNTER;
BEGIN
  ...
  IF SUM > LIMIT;
  RETURN;
  SUM = SUM + 1;
  ...
END

- 异常终止将在后面讨论
SUBROUTINE-DEFINITION

---

- a subroutine may be defined with
  FORMAL PARAMETERS

procedure  -  PROC EXAMPLE'PROC (P1, P2 : P3, P4);
  BEGIN
  "DECLARATIONS - PARAMETERS AND OTHERS"
  "EXECUTION"
  END

function    -  PROC EXAMPLE'FUNC (P1, P2 : P3) U 15;
  BEGIN
  "DECLARATIONS - PARAMETERS AND OTHERS"
  "EXECUTION"
  END
FORMAL PARAMETERS

- formal parameters may be:

  input only
  output only
  input and output

- formal parameters may be:

  input  - items, tables, blocks
          | labels, subroutines
  output - items, tables, blocks

- the value of a formal input parameter may not be changed

- output parameters may be used as input values
PROCEDURE-DEFINITION

PROC EXAMPLE'PROC (IN'P1 : OUT'P1);
BEGIN
ITEM IN'P1 F;
ITEM OUT'P1 F;
;
END

1 input parameter
1 output parameter
both parameters must be declared
PROCEDURE-DEFINITION

---------------------------------------------------------------

PROC EXAMPLE'PROC (IN'P1 : OUT'P1);
   BEGIN
   TYPE SINGLE'FLOAT F;
   ITEM IN'P1 SINGLE'FLOAT;
   ITEM OUT'P1 SINGLE'FLOAT;
   :
   :
   END

all parameters and type-names used by those
parameters must be declared
PROC EXAMPLE'FUNC (IN'P1) F;
BEGIN
ITEM IN'P1 F;
:
:
END

1 input parameter
function returns a floating point result
PROC SET'UP;
  BEGIN
    ::
    END

PROC EVALUATE (: OUT'P1, OUT'P2);
  BEGIN
    ::
    END

PROC PRIME (NUMBER : FACTORTAB) B 1;
  BEGIN
    ::
    END
SUBROUTINE INVOCATION

procedure -

EXAMPLE'PROC (VALUE : ANSWER);

function -

ANSWER = EXAMPLE'FUNC (VALUE);

a subroutine is called with
ACTUAL PARAMETERS
ACTUAL PARAMETERS

- actual parameters may be:
  
  input only
  output only
  input and output

- actual parameters may be:

  input    items, tables, blocks, formulae
           labels, subroutines
  output   items, tables, blocks

- actual parameters of subroutine-call must match
  formal parameters of subroutine-definition in:

  input/output kind
  number
  type
PROCEDURE

TYPE SINGLE'FLOT F;
ITEM VALUE SINGLE'FLOT = 3.0;
ITEM ANSWER SINGLE'FLOT = 0.0;
::
:: EXAMPLE'PROC (VALUE : ANSWER);
::
:: PROC EXAMPLE'PROC (IN'P1 : OUT'P1);
   BEGIN
   "PROC"
   ITEM IN'P1 F;
   ITEM OUT'P1 F;
   OUT'P1 = IN'P1 ** IN'P1;
   END "PROC"
FUNCTION

TYPE SINGLE'FLOAT F;
ITEM VALUE SINGLE'FLOAT = 3.0;
ITEM ANSWER SINGLE'FLOAT = 0.0;
::
::
ANSWER = EXAMPLE'FUNC (VALUE);
::
::
PROC EXAMPLE'FUNC (IN’P1) F;
    BEGIN
        "PROC"
    ITEM IN’P1 F;
    EXAMPLE'FUNC = IN’P1 ** IN’P1;
    END
        "PROC"
PARAMETER BINDING

associates the value of actual parameter of subroutine call with formal parameter of subroutine-definition

- item input-actuals
  bound by value - copied in

- item output-actuals
  bound by value-result - copied in and out

- table and block input- and output-actuals
  bound by reference
  manipulate the actual parameter directly
SUBROUTINE USAGE

subroutine may be called 3 ways:

"regular"

recursively - subroutine calls itself directly
or indirectly

reentrantly - several "copies" may be executing
concurrently
PROC RFACTORIZATION REC (IN'ARG) U;
BEGIN  "PROC"
ITEM IN'ARG U;
IF IN'ARG <= 1;
    RFACTORIZATION = 1;
ELSE
    RFACTORIZATION = RFACTORIZATION (IN'ARG - 1) * IN'ARG;
END  "PROC"
RECURSION

called with IN’ARG = 4

1st call  ------ > RFACTORIAL (4)  ---- >
2nd call ------ >  
3rd call ------ >  
4th call ------ >  
SUBROUTINE TERMINATION

- normal  - execute last statement in subroutine
  - execute return-statement
  - value-result parameters copied out
  - reference parameters fully set
  - function return-value copied out
NORMAL SUBROUTINE TERMINATION

---

PROC MATCH (IN'KEY, IN'SEARCHTAB) B 1;
BEGIN            "PROC"
ITEM IN'KEY S;
TABLE IN'SEARCHTAB (1 : 10);
ITEM IN'SEARCH S;

FOR I : 1 BY 1 WHILE I <= 10;
  IF IN'SEARCH (I) = IN'KEY;
    BEGIN               "FOUND MATCH"
      MATCH = TRUE;
      RETURN;
    END                "FOUND MATCH"
  ELSE
    MATCH = FALSE;
  END            "PROC"
SUBROUTINE TERMINATION

- abnormal - execute abort-statement
  - execute stop-statement
  - execute goto-statement

- value-result parameters NOT copied out
- reference parameters partially set
- function return-value NOT copied out
ABORT

- like "signal-handling"
- used for error processing
ABORT

BEGIN

COMPUTE (COST : EFFORT) ABORT CHECKOUT;

CHECKOUT: OVER’LIMIT (COST);

END

PROC COMPUTE (IN’$$ : OUT’VALUE);
    BEGIN
        :
        :
        IF IN’$$ > LIMIT;
            ABORT;
        END
        OUT’VALUE = GET’TOTAL (IN’$$);
    END
ABORT

BEGIN

COMPUTE (COST : EFFORT) ABORT CHECKOUT;

CHECKOUT: OVER'LIMIT (COST);

END

PROC COMPUTE (IN'$$: OUT'VALUE);
    BEGIN
    :
    :
    GET'TOTAL (IN'$$);
    END
PROC GET'TOTAL (IN'MONEY);
    BEGIN
    :
    :
    ABORT;
    :
    END

- abort conditions are "propagated out"
SAMPLE PROGRAM 1

START
PROGRAM PERFECT'NUMBER;
BEGIN
CONSTANT ITEM NUMBER U = 28;
TABLE FACTOR'TAB (1 : NUMBER);
ITEM FACTOR B 1 = NUMBER (FALSE);
ITEM IS'PERFECT B 1 = FALSE;
ITEM TEST'SUM U = 0;
ITEM LOOP'LIMIT U = 0;

LOOP'LIMIT = (* S *) ((* F *) (NUMBER)) ** (0.5));

"SET UP TABLE OF FACTORS"

FIND'FACTORS (NUMBER, LOOP'LIMIT : FACTOR'TAB);

FOR I : 2 BY 1 WHILE I <= LOOP'LIMIT;  "SUM THE FACTORS"
  IF (FACTOR (I) = TRUE);
    TEST'SUM = TEST'SUM + I + (NUMBER / I);
  TEST'SUM = TEST'SUM + I;
  IF (TEST'SUM = NUMBER);
    IS'PERFECT = TRUE;  "TEST IF PERFECT"
SAMPLE PROGRAM 1

PROC FIND'FACTORS (IN'NUMBER, IN'LIMI T : OUT'FACTOR'TAB);
BEGIN
ITEM IN'NUMBER U;
ITEM IN'LIMI T U;
TABLE OUT'FACTOR'TAB (1 : 28);
ITEM OUT'FACTOR B I;
FOR I : 1 BY 1 WHILE I <= IN'LIMI T;
IF (IN'NUMBER MOD I = 0);
BEGIN
OUT'FACTOR (I) = TRUE;
OUT'FACTOR (IN'NUMBER / I) = TRUE;
END
END
TERM
START
PROGRAM CALL'MATADD;
    BEGIN    "PROGRAM"
    TYPE SINGLE'FLOAT F;
    CONSTANT ITEM MAT'LIMIT U = 3;
    TABLE MATRIX1 (1 : 3, 1 : 3) = 9 (5.0);
    ITEM MATRIX1'I SINGLE'FLOAT;
    TABLE MATRIX2 (1 : 3, 1 : 3) = 3 (1.0, 2.0, 3.0);
    ITEM MATRIX2'I SINGLE'FLOAT;
    TABLE MATRIX'ANSWER (1 : 3, 1 : 3);
    ITEM MATRIX'ANSWER'I SINGLE'FLOAT;
    
    MATRIX'ADD (MATRIX1, MATRIX2, MAT'LIMIT :
                 MATRIX'ANSWER);
*BOUND TABLES - SAMPLE PROGRAM 2

PROC MATRIX'ADD (IN'MAT1, IN'MAT2, IN'LIMIT : OUT'MAT);
BEGIN    "PROC"
  TABLE IN'MAT1 ( *, * );
    ITEM IN'MAT1'I F;
  TABLE IN'MAT2 ( *, * );
    ITEM IN'MAT2'I F;
  ITEM IN'LIMIT U;
  TABLE OUT'MAT ( *, * );
    ITEM OUT'MAT F;
  FOR I : 0 BY 1 WHILE I < IN'LIMIT;
    FOR J : 0 BY 1 WHILE J < IN'LIMIT;
      OUT'MAT'I ( I, J ) =
        IN'MAT1'I ( I, J ) +
        IN'MAT2'I ( I, J );
  END    "PROC"
END    "PROGRAM"
TERM
MODULES AND EXTERNALS
PROGRAM ORGANIZATION

main-program-module

all declarations
all executable code
all subroutines

complete program

main-program-module

some declarations
some executable code

compool-module

some declarations

compool-module

some declarations

procedure-module

some subroutines

procedure-module

some subroutines

complete program
DECLARATIONS

- non-executable

- declare a name and attributes associated with that name

- all data names must be declared
# DATA STORAGE

<table>
<thead>
<tr>
<th>STATIC</th>
<th>AUTOMATIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>allocated at beginning of program</td>
<td>allocated when subroutines are invoked</td>
</tr>
<tr>
<td>deallocated at end of program</td>
<td>deallocated when subroutines terminate</td>
</tr>
<tr>
<td>data objects not declared in subroutines or with STATIC or constants</td>
<td>variables declared in subroutines</td>
</tr>
</tbody>
</table>
DATA STORAGE

ITEM XX STATIC U;
ITEM F23 STATIC F 23 = 17E1;
TABLE TAB1 STATIC (1 : 3);
ITEM ITEM1 B 24;

- only STATIC data may be preset
SCOPE

scope of a declaration
the area in which it applies

in J73, scope => subroutine (program)
from the point a name is declared to the end
of the subroutine (program)
PROC P1
BEGIN
ITEM AA
ITEM BB
ITEM CC
END

PROC P2
BEGIN
ITEM ZZ
TABLE XY
END

- Scope is like a one-way mirror looking out...

- declared and usable in P1
- declared and usable in P2
- also available in P2

- P1
- P2
- AA
- BB
- CC
PROC P1

BEGIN
ITEM AA S 15;
ITEM BB
ITEM CC
:
PROC P2

BEGIN
ITEM AA F 23;
ITEM ZZ
TABLE YY
:
END

END

integer AA has effect here

floating point AA has effect here; integer AA is not directly accessible
SCOPE

names "visible"
"information hiding"

PROC P1
BEGIN
ITEM AA
ITEM BB
TABLE ZZ
:----------------------------- -> P1,P2,P4
PROC P2
BEGIN
ITEM HH
:------------------------- ---- -> P2,P3,P4
PROC P3
BEGIN
TABLE II
:-------------------------- -> P3
ITEM II
:------------------------- -> P4
PROC P4
BEGIN
ITEM ZZ
ITEM YY
:------------------------- -> P1,AA,BB
END

139
COMPLETE PROGRAM

START PROGRAM
TERM
main-program-module
must have one and only one

START COMPOOL
TERM
compool-module
zero or more

START DEF PROC
TERM
procedure-module
zero or more

START-TERM are module (compilation unit) delimiters
MAIN-PROGRAM-MODULE

START
PROGRAM name;
BEGIN
"DECLARATIONS"
"EXECUTION"
"SUBROUTINES"
END
"NON-NESTED SUBROUTINES"
TERM
COMPLETE PROGRAM

---

START
PROGRAM SAMPLE;
BEGIN
ITEM CALLED B 1;
ITEM VALUE S = 10;
SUM (: VALUE);
CALLED = TRUE;

PROC SUM ( OUT);
BEGIN
ITEM OUT S;
OUT = OUT + 1;
END

END
TERM
PROCEDURE-MODULE

START
"DECLARATIONS"    "STATIC"
"SUBROUTINE-DEFINITIONS"
TERM

START
PROGRAM PP;
: TERM

<----
START
PROC P1;
PROC P2;
: TERM
EXTERNAL-DECLARATIONS

DEF exports name and attributes
     makes name available for access by other modules

REF imports name and attributes
     references name declared external elsewhere

used on data-names and subroutine-names
EXTERNAL-DECLARATIONS

START
PROGRAM SAMPLE;
BEGIN
REF PROC SUM (: OUT);
BEGIN
ITEM OUT S;
END
REF ITEM CALLED B 1;
ITEM VALUE S = 10;
SUM (: VALUE);
CALLED = TRUE;
END
TERM

arrow external
subroutine-declaration
arrow external item-declaration
arrow call to external subroutine
arrow reference of external item
EXTERNAL-DECLARATIONS

START
DEF ITEM CALLED B 1;
DEF PROC SUM (: OUT);
BEGIN
ITEM OUT S;
OUT = OUT + 1;
END
TERM

- DEF exports a name
- REF imports a name
- DEF and REF must MATCH
COMPOOL-MODULE

- "common declarations pool"
- declarations only; no executable code
- only reliable method for inter-module communication
- external-declarations, constants, type-declarations

START
COMPOOL_DECLS;
    DEF ITEM CALLED B 1;
    REF PROC SUM (: OUT);
    BEGIN
        ITEM OUT S;
    END

TERM
COMPOOL-DIRECTIVE

---

- imports all or selected information from a
  PREPROCESSED compool-file

START
!COMPOOL ('DECLS');
  :
  :
TERM
COMPLETE PROGRAM

START
COMPOOL DECLS;
  DEF ITEM CALLED B 1;
  REF PROC SUM (: OUT);
  BEGIN
  ITEM OUT S;
  END
TERM

compool-module

START
!COMPOOL ('DECLS');
DEF PROC SUM (: OUT);
  BEGIN
  ITEM OUT S;
  OUT = OUT + 1;
  END
TERM

procedure-module
COMPLETED PROGRAM

START
!COMPOOL ('DECLS');
PROGRAM SAMPLE;
  BEGIN
    ITEM VALUE S = 10;
    SUM (: VALUE);
    CALLED = TRUE;
  END
TERM

main-program-module
MODULE SCOPE

-----------------------------------------------

COMPOOL DECLS

system scope

-----------------------------------------------

COMPOOL DECLS

procedure-module
(DEF PROC SUM)
(REF PROC SUM)

system scope

-----------------------------------------------

COMPOOL DECLS

main-program-module
SAMPLE
(REF PROC SUM, DEF CALLED)

system scope
SAMPLE PROGRAM

START
COMPOOL TYPES;
   TYPE U'WORD U;
   TYPE SINGLE'FLOAT F 23;
TERM

START
!COMPOOL ('TYPES');
COMPOOL DATABASE;
   CONSTANT ITEM MAT'LIMIT U'WORD = 3;
   DEF TABLE MATRIX1 (1 : 3, 1 : 3) = 9 (5.0);
       ITEM MATRIX1'I SINGLE'FLOAT;
   DEF TABLE MATRIX2 (1 : 3, 1 : 3) = 3 (1.0, 2.0, 3.0);
       ITEM MATRIX2'I SINGLE'FLOAT;
   DEF TABLE MATRIX'ANSWER (1 : 3, 1 : 3);
       ITEM MATRIX'ANSWER SINGLE'FLOAT;
TERM
SAMPLE PROGRAM

START
!COMPOOL ('TYPES');
COMPOOL REFPROCS;
    REF PROC MATRIX'ADD (IN'MAT1, IN'MAT2, IN'LIMIT : OUT'MAT):
    BEGIN
        TABLE IN'MAT1 (*, *);
        ITEM IN'MAT1'I SINGLE'FLOAT;
        TABLE IN'MAT2 (*, *);
        ITEM IN'MAT2'I SINGLE'FLOAT;
        ITEM IN'LIMIT U'WORD;
        TABLE OUT'MAT (*, *);
        ITEM OUT'MAT'I SINGLE'FLOAT;
    END

TERM
SAMPLE PROGRAM

START
!COMPOOL ('TYPES');
!COMPOOL ('REFPROCS');
DEF PROC MATRIX'ADD (IN'MAT1, IN'MAT2, IN'LIMIT : OUT'MAT):
BEGIN
TABLE IN'MAT1 (*, *);
   ITEM IN'MAT1'I SINGLE'FLOAT;
TABLE IN'MAT2 (*, *);
   ITEM IN'MAT2'I SINGLE'FLOAT;
ITEM IN'LIMIT U'WORD;
TABLE OUT'MAT (*, *);
   ITEM OUT'MAT'I SINGLE'FLOAT;

FOR I : 0 BY 1 WHILE I < IN'LIMIT;
   FOR J : 0 BY 1 WHILE J < IN'LIMIT;
      OUT'MAT (I, J) = IN'MAT1'I (I, J) +
      IN'MAT2'I (I, J);
END

TERM
SAMPLE PROGRAM

START
!COMPOOL ('DATABASE');
!COMPOOL ('REFPROCS');
PROGRAM CALL'MATADD;
    BEGIN
        MATRIX'ADD (MATRIX1, MATRIX2, MAT'LIMIT :
                      MATRIX'ANSWER);
    END
TERM
ORDINARY TABLE

compiler determines how to position items
in a table

this is the default
TABLE DATA (1 : 20);
BEGIN
ITEM U3 U 3;
ITEM B1 B 1;
ITEM B3 B 3;
END

U3 (1)
B1 (1)
B3 (1)
U3 (2)
B1 (2)
B3 (2)
U3 (20)
B1 (20)
B3 (20)

20 entries
3 words / entry
60 words
PACKING

==============================================

describes how items within a single entry are allocated

N  - no packing; 1 item / word (default)

M  - medium packing; implementation-dependent

D  - dense packing; as many items (from only 1 entry) as possible
PACKING

---

TABLE DATA (1 : 20) D;
BEGIN
ITEM U3 U 3;
ITEM B1 B 1;
ITEM B3 B 3;
END

<table>
<thead>
<tr>
<th>U3(1)</th>
<th>B1(1)</th>
<th>B3(1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>U3(2)</td>
<td>B1(2)</td>
<td>B3(2)</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>U3(20)</td>
<td>B1(20)</td>
<td>B3(20)</td>
</tr>
</tbody>
</table>

ordinary
serial
entry-by-entry
dense packing
3 items / word

20 entries
1 word / entry
20 words
STRUCTURE

describes how entire entries are allocated

serial       - entry-by-entry
parallel     - first word of each entry
tight        - serial; more than one entry per word;
              densely packed by default
**STRUCTURE**

---

**TABLE DATA (1 : 20) PARALLEL;**

**BEGIN**

ITEM U3 U 3;
ITEM B1 B 1;
ITEM B3 B 3;
END

**ordinary**

parallel

**all first words**

** : **

**all last words**

**1 item / word**

| U3 (1) |
| : |
| U3 (20) |
| B1 (1) |
| : |
| B1 (20) |
| B3 (1) |
| : |
| B3 (20) |

20 entries

3 words / entry

60 words
STRUCTURE

TABLE DATA (1 : 20) T 8;
BEGIN
ITEM U3 X 3;
ITEM B1 1;
ITEM B3 3;
END

<table>
<thead>
<tr>
<th>U3 (1)</th>
<th>B1 (1)</th>
<th>B3 (1)</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>U3 (2)</td>
<td>B1 (2)</td>
<td>B3 (2)</td>
<td>X</td>
</tr>
<tr>
<td>U3 (3)</td>
<td>B1 (3)</td>
<td>B3 (3)</td>
<td>X</td>
</tr>
<tr>
<td>U3 (4)</td>
<td>B1 (4)</td>
<td>B3 (4)</td>
<td>X</td>
</tr>
</tbody>
</table>

3 items / halfword
2 entries / word
10 words
ORDINARY TABLES

time-space trade-off for medium and dense packed tables

time-space trade-off for tight structured tables

data references and presets same as previously seen
SPECIFIED TABLE

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>each item explicitly positioned by the programmer</td>
</tr>
<tr>
<td>no packing</td>
</tr>
<tr>
<td>items may share storage</td>
</tr>
<tr>
<td>fixed-length-entry or variable-length-entry</td>
</tr>
<tr>
<td>used to overlay data structures</td>
</tr>
<tr>
<td>used to interface with a peripheral</td>
</tr>
</tbody>
</table>
FIXED-LENGTH

---

TABLE INFO (1 : 10) W 3;
BEGIN
ITEM NAME C 2 POS (0, 0);
ITEM INITIAL C 1 POS (0, 0);
ITEM AGE S 7 POS (0, 1);
ITEM SCHOOL C 1 POS (8, 1);
ITEM RANK B 4 POS (12, 2);
END

<table>
<thead>
<tr>
<th>bit 0</th>
<th>8</th>
<th>12</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;------ NAME (1) -------&gt;</td>
<td>INITIAL (1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AGE (1)</td>
<td>SCHOOL (1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RANK (1)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| | <------ NAME (10) -------> | INITIAL (10) |
| AGE (10) | SCHOOL (10) |
| RANK (10) |

3 words / entry
SPECIFIED TABLES

again - time-space trade-off

data references as previously seen

presets as previously seen but -
one location may not be preset more than one time
OVERLAY-DECLARATION

purposes:

allocate data to share storage
allocate data at a specific address
allocate data in a given order
any combination

169
OVERLAY-DECLARATION

share storage: OVERLAY AA : BB;
specific address: OVERLAY POS (892) : CC;
specific order: OVERLAY DD, EE, FF;
OVERLAY-DECLARATION

TABLE LONG'FLOAT (1 : 10);
   ITEM I'LONG'FLOAT F 39;

TABLE SHORT'FLOAT (1 : 10);
   BEGIN
      ITEM I'SHORT'FLOAT F 23;
      ITEM EXCESS B 8;
   END

OVERLAY LONG'FLOAT : SHORT'FLOAT;

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>I'LONG'FLOAT (1)</td>
<td>I'LONG'FLOAT (2)</td>
</tr>
<tr>
<td>EXCESS (1)</td>
<td>I'SHORT'FLOAT (2)</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>I'LONG'FLOAT (2)</td>
<td>I'SHORT'FLOAT (2)</td>
</tr>
<tr>
<td>EXCESS (2)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
OVERLAY-DECLARATION

Table bitstrings (1 : 4);
  item 'bitstrings b 16;

item alpha u;

item beta s;

item gamma c 2;

Overlay bitstrings : gamma, w 1 (alpha : beta);

<table>
<thead>
<tr>
<th>'bitstring (1)</th>
<th>gamma</th>
</tr>
</thead>
<tbody>
<tr>
<td>'bitstring (2)</td>
<td></td>
</tr>
<tr>
<td>'bitstring (3)</td>
<td>alpha</td>
</tr>
<tr>
<td>'bitstring (4)</td>
<td>beta</td>
</tr>
</tbody>
</table>
DEFINE

define-declaration associates define-name with a string of text

DEFINE PI "3.1415927";

define-call causes the textual substitution to occur

(J73 code): AREA = PI * (RADIUS ** 2);
(expanded): AREA = 3.1415927 * (RADIUS ** 2);
DEFINE

- A define may be declared (and called) with parameters

```
DEFINE VECEQI (A, B) "!A (1) = !B (1);
    !A (2) = !B (2);
    !A (3) = !B (3)";
```

(J73 code): VECEQI (TARGET'VECTOR, SOURCE'VECTOR);

(expanded): TARGET'VECTOR (1) = SOURCE'VECTOR (1);
TARGET'VECTOR (2) = SOURCE'VECTOR (2);
TARGET'VECTOR (3) = SOURCE'VECTOR (3);
DEFINE

- a define may be used to produce complete declarations

DEFINE MATRIX (A, B, C)
  "TABLE !A (1 : 3, 1 : 3) !C;
  ITEM !A'1!B";

(J73 code): DEF MATRIX (MATRIX1, SINGLE'FLOAT,
"= 9 (5.0));
DEF MATRIX (MATRIX2, SINGLE'FLOAT,
"= 3 (1.0, 2.0, 3.0)));
DEF MATRIX (MATRIX'ANSWER, SINGLE'FLOAT);

(expanded): see COMPOOL DATABASE ...
DEFINE

- defines may be nested

DEFINE DIMENSIONS "(1 : 3, 1 : 3)";

DEFINE MATRIX (A, B, C)
  "TABLE !A DIMENSIONS !C;
  ITEM !A\'1 !B";

(J73 code):    DEF MATRIX (MATRIX1, SINGLE'FLOAT,
                  "= 9 (5.0)"");

(first expansion):
    DEF TABLE MATRIX1 DIMENSIONS = 9 (5.0);
    ITEM MATRIX1'1 SINGLE'FLOAT;

(second expansion):
    DEF TABLE MATRIX1 (1 : 3, 1 : 3) = 9 (5.0);
    ITEM MATRIX1'1 SINGLE'FLOAT;
NEGONE = SCN (-3.2)

\[ \text{NEGONE} = \begin{cases} 1 & \text{positive} \\ 0 & \text{zero} \\ -1 & \text{negative} \end{cases} \]

\( \text{returns sign of its argument SCN} \)

\[ \text{PREN} = \text{ABS} (5.0 - 15.0) \]
\[ \text{TEN} = \text{ABS} (-10) \]

\( \text{returns absolute value of its argument} \)

\[ \text{IM} = \text{VALUE} \oplus \text{LOC (TAB1)} \]

\( \text{returns machine-address of its argument} \)

===============================

BUILT-IN FUNCTIONS
# BUILT-IN FUNCTIONS

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT</td>
<td>Select substring of bits</td>
<td>B5 = BIT (B10, 2, 5);</td>
</tr>
<tr>
<td></td>
<td>Pseudo-variable; assign to substring of bits</td>
<td>BIT (B16, 0, 2) = B2;</td>
</tr>
<tr>
<td>BYTE</td>
<td>Select substring of characters</td>
<td>C4 = BYTE (C10, 3, 4);</td>
</tr>
<tr>
<td></td>
<td>Pseudo-variable; assign to substring of characters</td>
<td>BYTE (C5, 0, 1) = C1;</td>
</tr>
<tr>
<td>REP</td>
<td>Returns machine representation of its argument</td>
<td>B16 = REP (SPEED);</td>
</tr>
<tr>
<td></td>
<td>Pseudo-variable; change machine representation of its argument</td>
<td>REP (C1) = B8;</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>SHIFTL</td>
<td>logical left shift of bit string</td>
<td></td>
</tr>
<tr>
<td>SHIFTR</td>
<td>logical right shift of bit string</td>
<td></td>
</tr>
<tr>
<td>BITSIZE</td>
<td>returns number of bits allocated</td>
<td></td>
</tr>
<tr>
<td>BYTESIZE</td>
<td>returns number of bytes allocated</td>
<td></td>
</tr>
<tr>
<td>WORDSIZE</td>
<td>returns number of words allocated</td>
<td></td>
</tr>
</tbody>
</table>

- B110 = SHIFTL (B111);  
- B001 = SHIFTR (B011);  
- SIXTEEN = BITSIZE (S15);  
- TWO = BYTESIZE (S15);  
- ONE = WORDSIZE (S15);
BUILT-IN FUNCTIONS

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>NWDSSEN</td>
<td>returns number of words in a table-entry</td>
<td>TABLE TAB1; BEGIN ITEM CC C 4; ITEM UU U; END THREE = NWDSSEN (TAB1);</td>
</tr>
<tr>
<td>FIRST</td>
<td>returns highest-valued status-constant</td>
<td>TYPE LETTER STATUS ( V(A), V(B), V(C) ); AA = FIRST (LETTER);</td>
</tr>
<tr>
<td>LAST</td>
<td>returns lowest-valued status-constant</td>
<td>TYPE LETTER STATUS ( V(A), V(B), V(C) ); CC = LAST (LETTER);</td>
</tr>
</tbody>
</table>
BUILT-IN FUNCTIONS

NEXT returns designated predecessor/successor of status value
ITEM WXYZ STATUS
( V(W), V(X), V(Y), V(Z) )
= V(X);
WW = NEXT (WXYZ, -1);
ZZ = NEXT (WXYZ, 2);

NEXT returns incremented value of pointer argument
PTRPLUS2 = NEXT (PTR, 2);

UBOUND returns upper-bound of designated table dimension
TABLE TAB1 (1 : 3, 2 : 7);
ITEM ITM1 S;
SEVEN = UBOUND (TAB1, 1);

LBOUND returns lower-bound of designated table dimension
TABLE TAB1 (1 : 3, 2 : 7);
ITEM ITM1 S;
ONE = UBOUND (TAB1, 0);
DIRECTIVES

- module linkage
  !COMPOOL
  !LINKAGE

- optimization
  !LEFTRIGHT
  !REARRANGE
  !ORDER
  !INTERFERENCE
  !REDUCIBLE

- register control
  !BASE
  !ISBASE
  !DROP
DIRECTIVES

- text and listing
  !COPY
  !SKIP
  !BEGIN
  !END
  !LIST
  !NOLIST
  !EJECT

- miscellaneous
  !TRACE
  !INITIALIZE
  implementation-specific directives
Topics not covered:
(or covered very briefly ...)

=================================================

Pointers, dereference, pointer-qualified references
Blocks
Specified status-lists
Labels and subroutines as parameters
Implementation-parameters
Built-in functions
Table layout
Overlay
Compiler directives
Define capability
JOVIAL (J73) DOCUMENTATION

MIL-STD-1589A
MIL-STD-1589B
JOVIAL (J73) Computer Programming Manual
JOVIAL (J73) Course Notes
JOVIAL (J73) Video Course
JOVIAL (J73) Primer