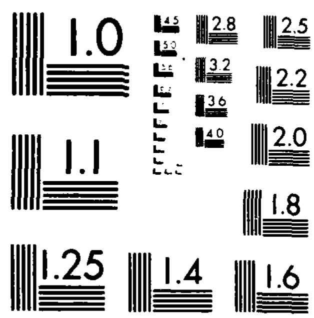
AD-A142 780 PROCEEDINGS PAPERS OF THE AFSC (AIR FORCE SYSTEMS COMMAND) AVIONICS STAND..(U) AERONAUTICAL SYSTEMS DIV WRIGHT-PATTERSON AFB OH DIRECTORATE O.. 1/3

UNCLASSIFIED C A PORUBCANSKY NOV 82 F/G 9/2 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963 A

ASD(ENA)-TR-82-5031
VOLUME V

AD-A142 780

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

SPONSORED BY
AIR FORCE SYSTEMS COMMAND

HOSTED BY

TUTORIAL

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

DTIC FILE COPY

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
Vice Chairman
2nd AFSC Standardization Conference

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

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER ASD(ENA)-TR-82-5031, VOLUME IV	2. GOVT ACCESSION NO. AD-A142 780	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Proceedings Papers of the Second AFSC Avionics Standardization Conference		5. TYPE OF REPORT & PERIOD COVERED Final Report 29 November - 2 December 1982
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Editor: Cynthia A. Porubcansky		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS HQ ASD/ENAS Wright-Patterson AFB OH 45433		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS HQ ASD/ENA Wright-Patterson AFB OH 45433		12. REPORT DATE November 1982
		13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Same as Above		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION DOWNGRADING SCHEDULE

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)

Computer Instruction Set Architecture, Multiplexing, Compilers, Support Software, Data Bus, Rational Standardization, Digital Avionics, System Integration, Stores Interface, Standardization, MIL-STD-1553, MIL-STD-1589 (JOVIAL), MIL-STD-1750, MIL-STD-1760, MIL-STD-1815 (ADA), MIL-STD-1862 (NEBULA).

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.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

This is Volume 5

Volume 1	Proceedings pp. 1-560
Volume 2	Proceedings pp. 561-1131
Volume 3	Governing Documents
Volume 4	MIL-STD-1553 Tutorial
Volume 5	MIL-STD-1589 Tutorial
Volume 6	MIL-STD-1679 Tutorial
Volume 7	MIL-STD-1750 Tutorial
Volume 8	MIL-STD-1815 Tutorial
Volume 9	Navy Case Study Tutorial

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 UNGLAMOROUS 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

DEPARTMENT OF THE AIR FORCE
HEADQUARTERS AIR FORCE SYSTEMS COMMAND
ANDREWS AIR FORCE BASE DC 20334

23 AUG 1982

REPLY TO ATTN OF: CV

SUBJECT: Second AFSC Standardization Conference

TO: ASD/CC

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

PRECEDING PAGE BLANK-NOT FILMED

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 co-ordinated 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 currrently 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!

INTRODUCTION

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?

:

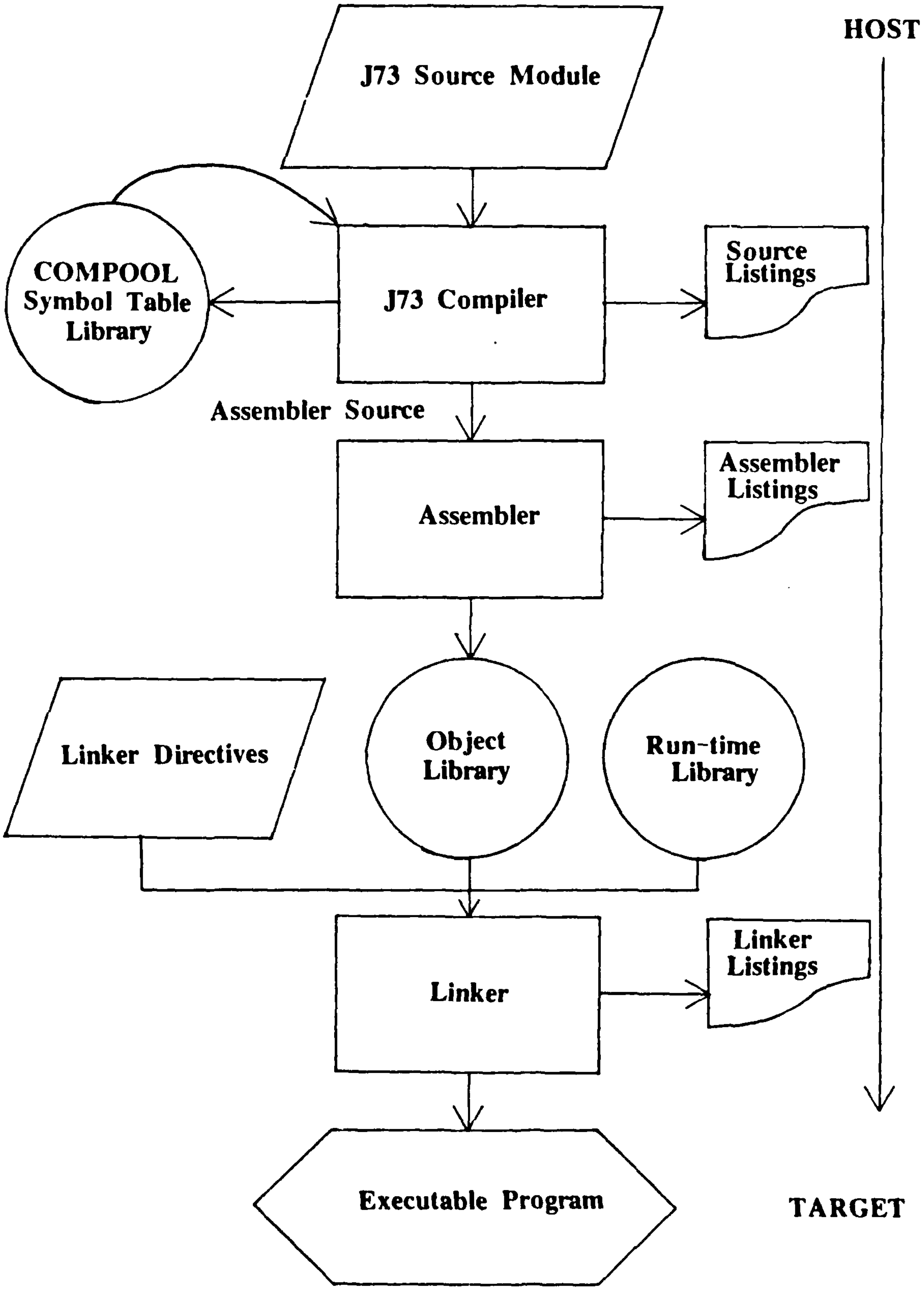
JOVIAL (J73) CAPABILITIES

- 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

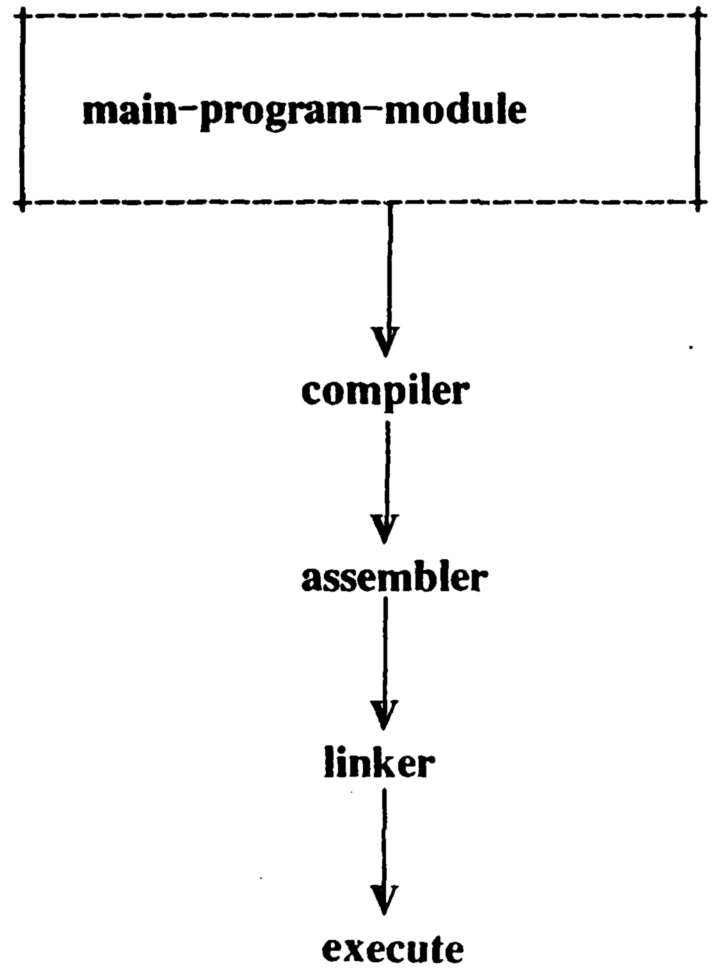
JOVIAL (J73) CAPABILITIES

- 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
  - !LINKAGE
  - !TRACE

PROGRAM ORGANIZATION


HOST
J73 Source Module
COMPOOL Symbol Table Library
J73 Compiler
Source Listings
Assembler Source
Assembler
Assembler Listings
Linker Directives
Object Library
Run-time Library
Linker
Linker Listings
Executable Program
TARGET

SAMPLE PROGRAM 1



SAMPLE PROGRAM 1

```
==================================================================

START

PROGRAM COUNTER;

     BEGIN          "MAIN-PROGRAM-MODULE"

     "DECLARATIONS"

     "EXECUTION"

     "SUBROUTINES"

     END            "MAIN-PROGRAM-MODULE"

TERM
```

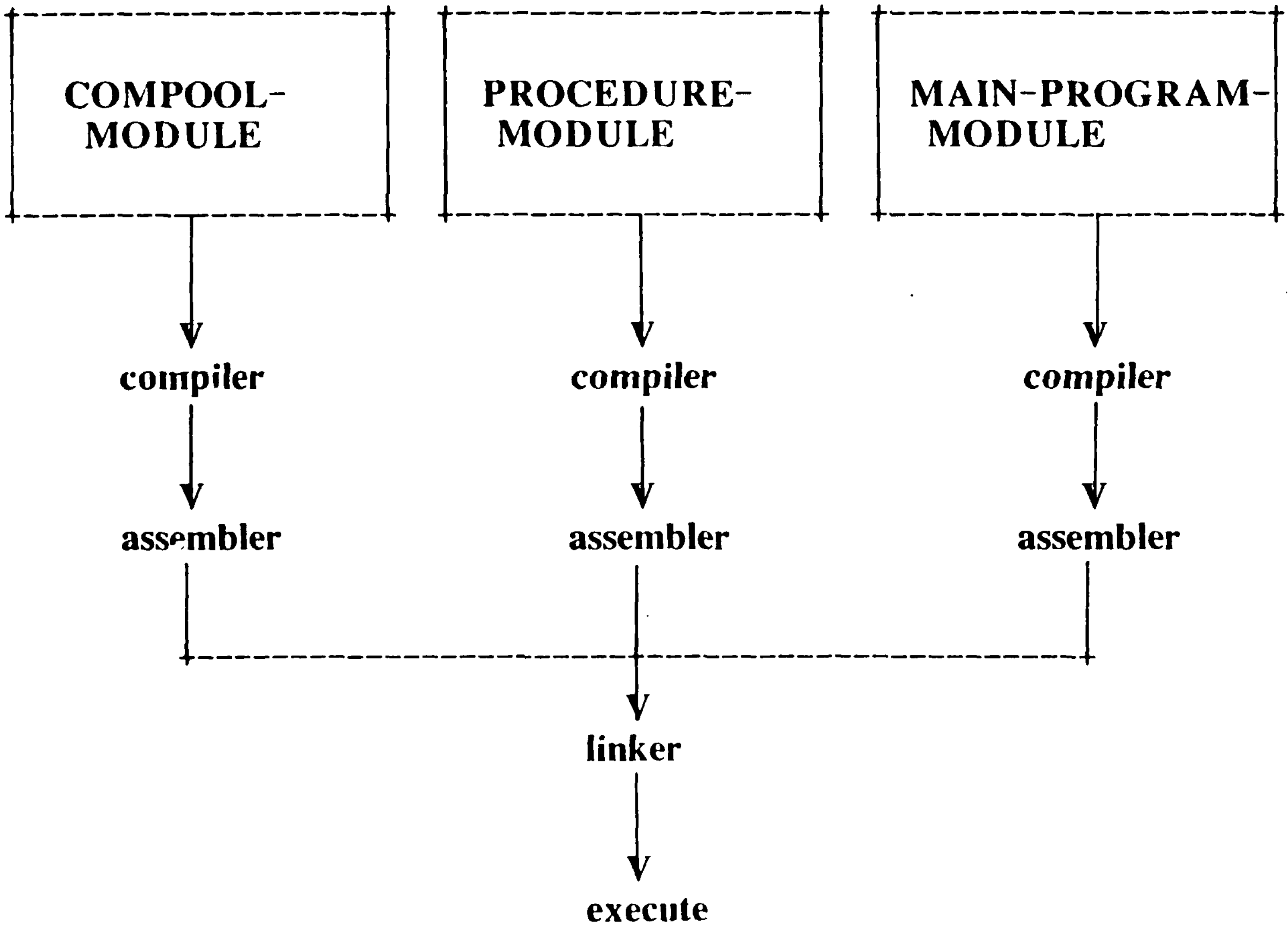
SAMPLE PROGRAM 1

```
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



SAMPLE PROGRAM 2

```
START
COMPOOL  DECLS;
:
:
TERM
```

compool-module

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

procedure-module

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

main-program-module

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
```

compool-module

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

TERM
```

procedure-module

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

main-program-module

DATA-DECLARATIONS

J73 DATA OBJECTS

ITEM	simplest data object
TABLE	array, record, or array of records of ITEMs
BLOCK	groupof ITEMs, TABLEs, and/or other BLOCKs

- 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 immediated value in assembly code and not allocated preset to its constant value referenced only

I T E M S

DATA TYPES

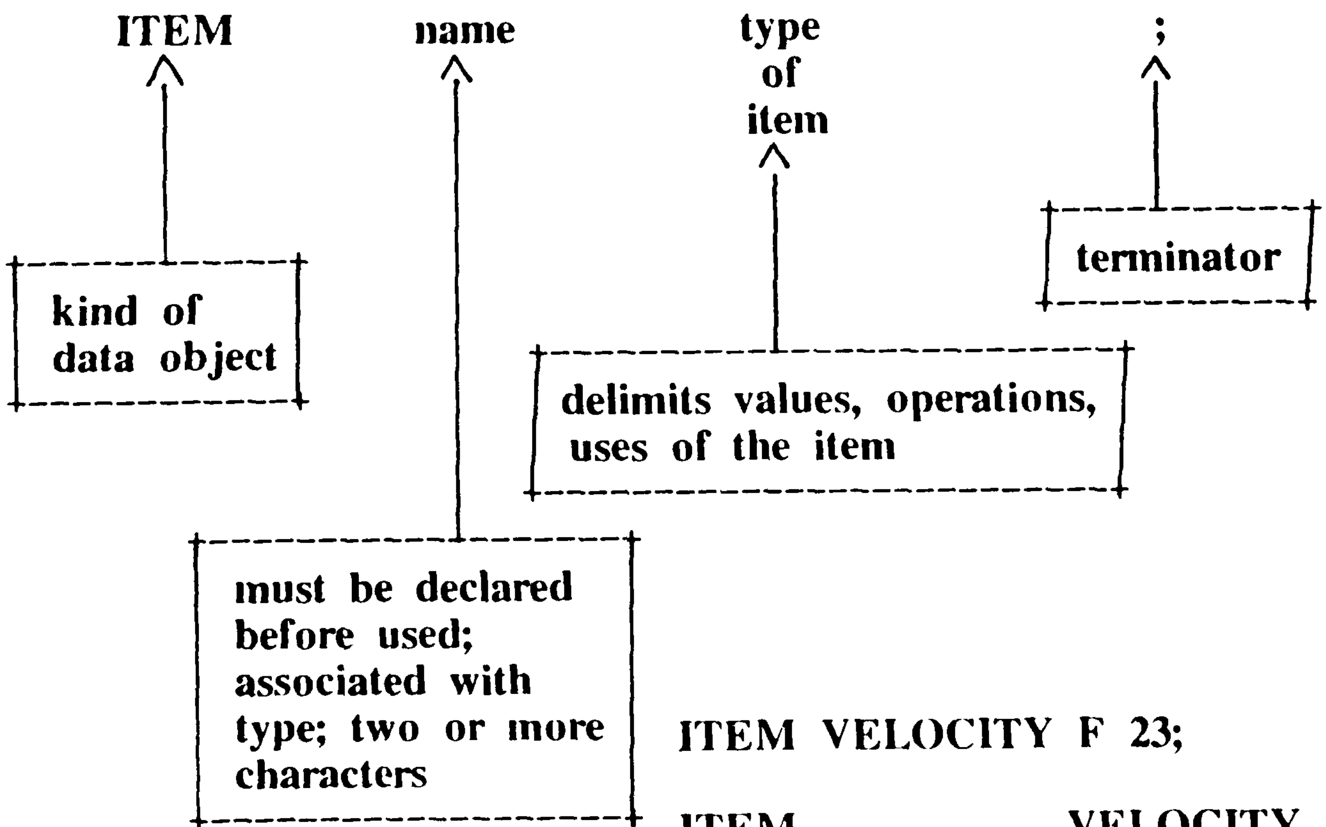
U	unsigned integer	0 --> largest positive whole number
S	signed integer	smallest negative --> 0 --> largest positive whole number
F	floating point	fractional representation
A	fixed point	fractional representation - may use integer arithmetic
B	bit	
C	character	
STATUS	status (enumeration)	
P	pointer (access)	

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 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'0001111000';

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;   }  same type

TYPE COUNTER'TYPE U;

ITEM I'LOOP COUNTER'TYPE;
ITEM J'LOOP COUNTER'TYPE = 3;
CONSTANT ITEM TEST$$LOOP COUNTER'TYPE = 7;
```

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.

EQUIVALENT types:	IMPLICITLY CONVERTIBLE types:
U 3 = U 3	U 10 = U 8
S 24 = S 24	U 2 = U 15
	S 12 = U 10
	U 7 = S 20
F 7 = F 7	F 23 = F 3
A 10, 2 = A 10, 2	A 6, 7 = A 3, 2
B 8 = B 8	B 10 = B 2
	B 8 = B 64
C 3 = C 3	C 2 = C 9
	C 7 = C 3

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.

EXPLICIT CONVERSION

```
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

<------------------ TARGET (to) type ------------------>

SOURCE (from) type	U short	U long	S short	S long	F short	F long	A short	A long	B short	B long	C short	C long
U short	E	I	I	I	X	X	X	X	X	X		
U long	I	E	I	I	X	X	X	X	X	X		
S short	I	I	E	I	X	X	X	X	X	X		
S long	I	I	I	E	X	X	X	X	X	X		
F short	X	X	X	X	E	I	X	X	X	X		
F long	X	X	X	X	I	E	X	X	X	X		
A short	X	X	X	X	X	X	E	I	X	X		
A long	X	X	X	X	X	X	I	E	X	X		
B short	*	*	*	*	*	*	*	*	E	I	X	X
B long	*	*	*	*	*	*	*	*	I	E	X	X
C short									X	X	E	I
C long									X	X	I	E

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	AND	logical "and"
(bit)	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.

FORMULAE

```
ITEM B5 B 5 = 1B'10101';
ITEM B10 B 10 = 1B'1111100000';
ITEM B10'TOO B 10 = 1B'1010101010';
ITEM BOOLEANT B 1 = TRUE;
ITEM BOOLEANF B 1 = FALSE;
ITEM RESULT1 B 1;
ITEM RESULT2 B 10;

RESULT1 = BOOLEANT AND BOOLEANF;        " 1B'0' "

RESULT1 = BOOLEANT OR 1B'1';             " 1B'1' "

RESULT2 = B10 EQV B10'TOO;               " 1B'1010110101' "

RESULT2 = (B10 XOR B10'TOO) AND B5;     " 1B'0000000000' "

RESULT1 = NOT BOOLEANT;                  " 1B'0' "

RESULT1 = BOOLEANF 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

EXECUTABLE STATEMENTS

EXECUTABLE-STATEMENTS

- assignment-statement
- conditional statements
  - if-statement
  - case-statement
- loop-statements
  - while-loop
  - for-loop
- transfer of control
  - exit-statement
  - goto-statement

ASSIGNMENT-STATEMENT

==

```
ITEM ANSWER1 S 15;
ITEM ANSWER2 S 15;

ANSWER1 = 67;

ANSWER2 = (-4[illegible] (-12)[illegible]

ANSWER1 = A[illegible]

ANSWER1, ANSWER[illegible]
```

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,	if SUM = 300.0,
ANSWER = 250.0	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 LETER 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 LETER 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 preceeding 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               "CASE"
     (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

```
if CATEGORY = 'G',      COUNTER = 0
                        NUMBER = 0

if CATEGORY = 'B',      COUNTER = 9
                        NUMBER = 3

if CATEGORY = 'C',      COUNTER = 8
                        NUMBER = 2

if CATEGORY = 'E',      COUNTER = 5
                        NUMBER = 2
```

LOOP-STATEMENTS

==

repeated execution of a controlled-statement

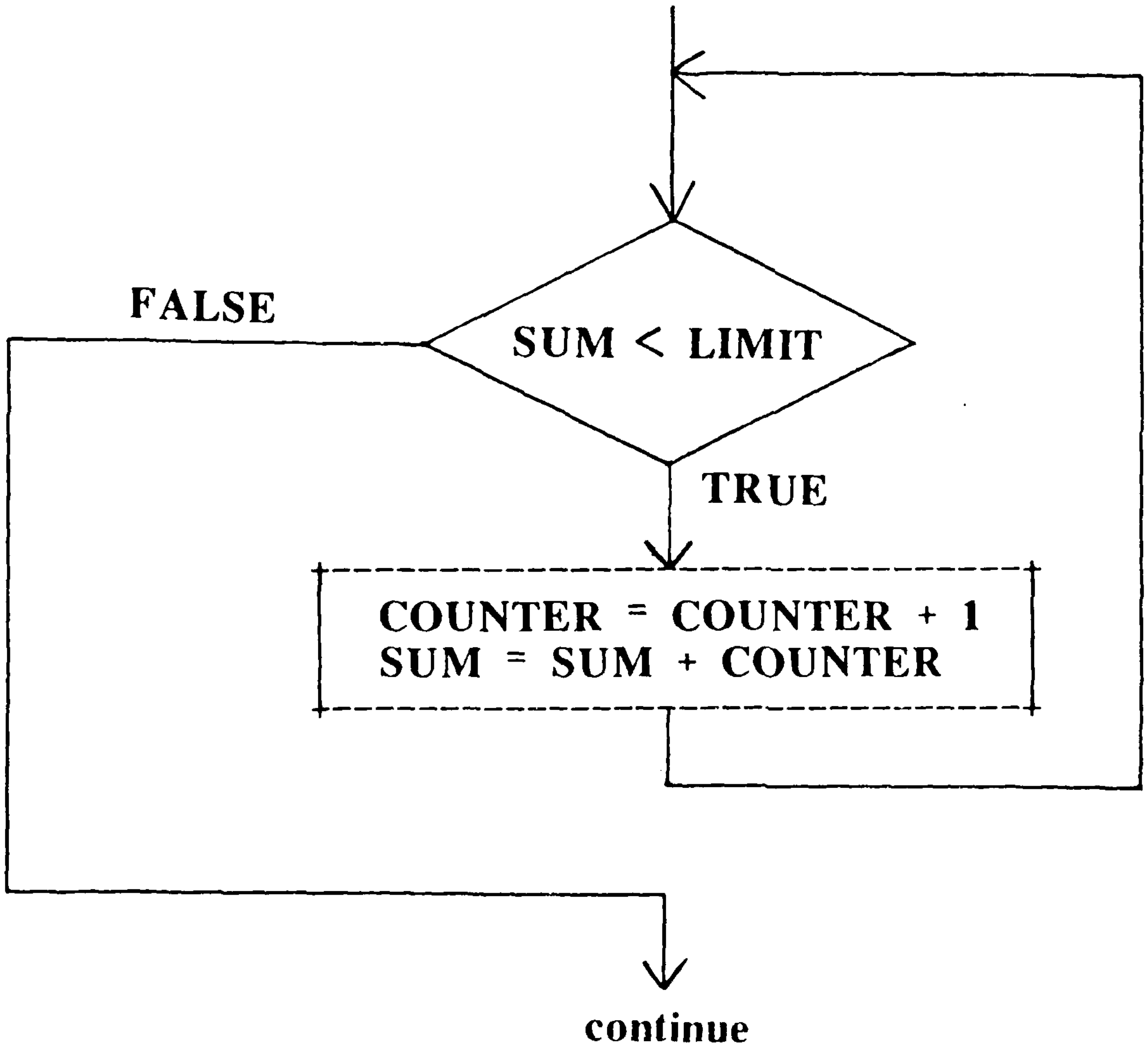
two kinds of loops:

- 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

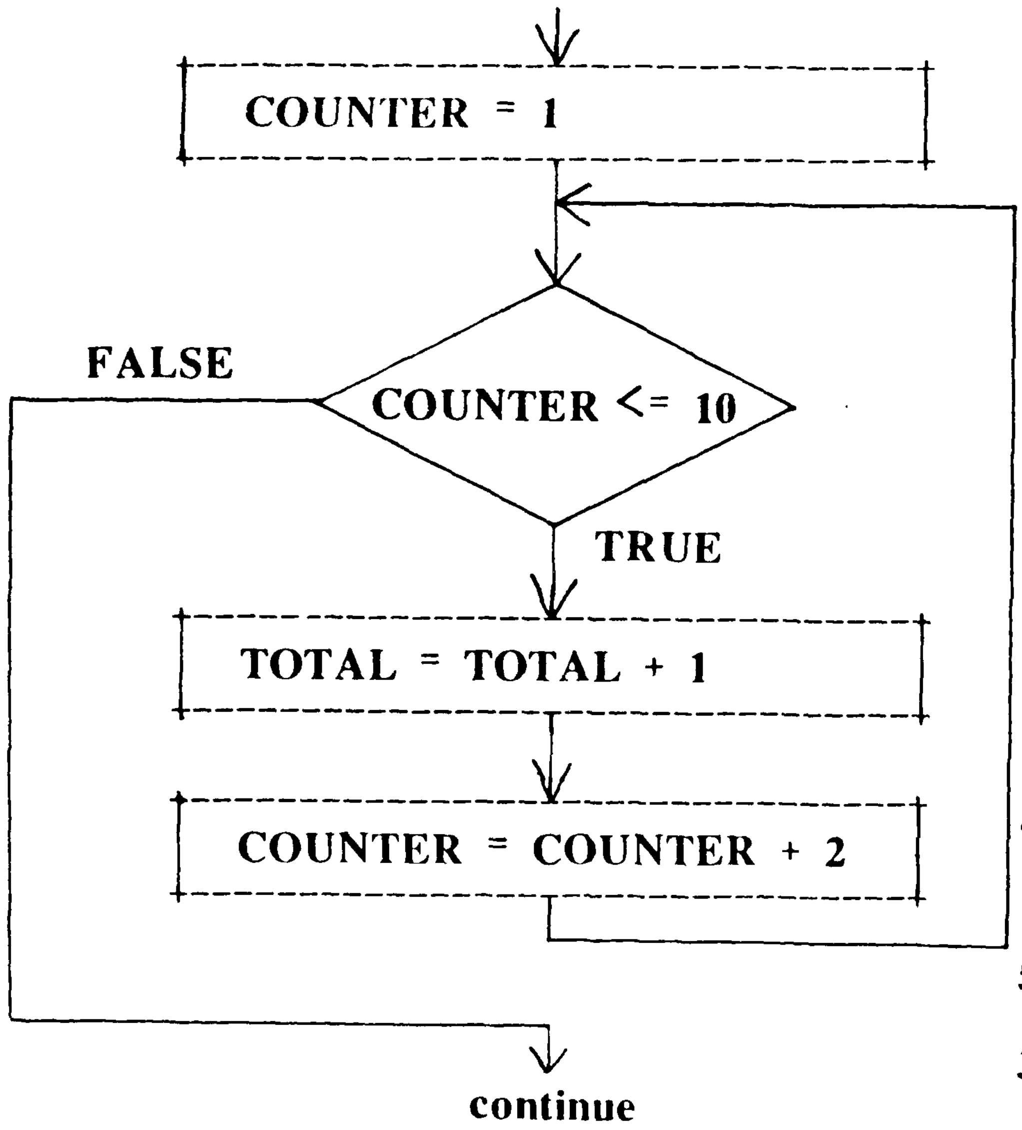
3) loop

4) if FALSE, exit loop

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

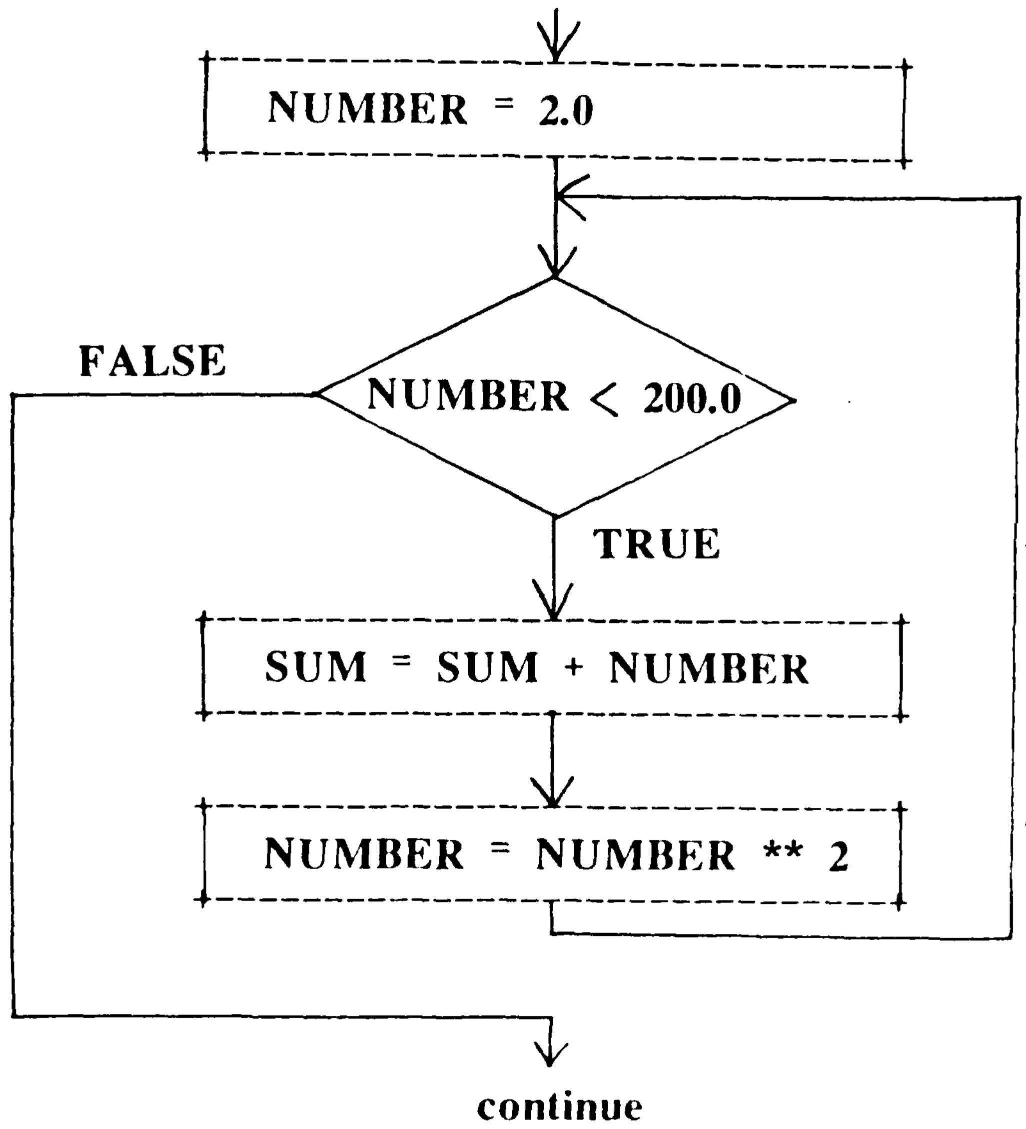
FOR-LOOP

```
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

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
```

T A B L E S

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'I F 23;
```

TABLES

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

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;            1 entry
     ITEM HISTORY C 1;            6 items / entry
     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');           TABLE GRADES = 2 ('A', 'B');
     BEGIN                             BEGIN
     :                                 :
     END                               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 RITING 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 VECTOR (1 : 20)                20 entries
    ITEM VECTOR'I F 23;               1 item / entry

TABLE MATRIX (1 : 4, 1 : 4);         16 entries
    ITEM MATRIX'I S 15;               1 item / entry
```

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'I (3) = 32E7;

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

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


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

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


FOR I : 1 BY 1 WHILE I <= 4;
     FOR J : 1 BY 1 WHILE J <= 4;
          MATRIX (I, J) = 0;
```

TABLE-DECLARATION

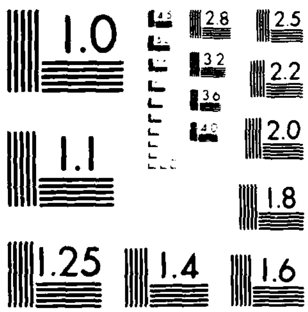
tables may be preset ...

```
TABLE MATRIX (1 : 4, 1 : 4);
     ITEM MATRIX'I S 15 = 4 (1, 0, 0, 0);

TABLE MATRIX (1 : 4, 1 : 4) = 4(1, 0, 0, 0);
     ITEM MATRIX'I S 15;

TABLE MATRIX (1 : 4, 1 : 4);
     ITEM MATRIX'I 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);
```

1,1	1,2	1,3	1,4
2,1	2,2	2,3	2,4
3,1	3,2	3,3	3,4
4,1	4,2	4,3	4,4

AD-A142 780 PROCEEDINGS PAPERS OF THE AFSC (AIR FORCE SYSTEMS COMMAND) AVIONICS STAND..(U) AERONAUTICAL SYSTEMS DIV WRIGHT-PATTERSON AFB OH DIRECTORATE O.. 2/3

UNCLASSIFIED C A PORUBCANSKY NOV 82 F/G 9/2 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963 A

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;                              9 entries
     ITEM HEIGHT F 23;                             4 items / entry
     END

TABLE CLASS'RECORD (1 : 5, 1 : 4);
     BEGIN                                        20 entries
     ITEM STUDENT'NAME C 20;                       3 items / entry
     ITEM AGE S;
     ITEM SEX STATUS ( V(MALE), V(FEMALE) );
     END
```

SUBSCRIPTS

==

```
LENGTH (9) = 14.5;

WIDTH (4) = HEIGHT (I);

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);
```

TABLE-DECLARATION

==

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

can type - 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 tc 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         "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 I : 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"
TERM
```

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"
    END                        "PROGRAM"
TERM
```

SUBROUTINES

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

===

normal – execute last statement in subroutine

– execute return-statement

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

– abnormal termination is discussed later

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

FUNCTION-DEFINITION

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

1 input parameter
function returns a floating point result

SUBROUTINE-DEFINITION

```
===============================================================

PROC SET'UP;                                    procedure
    BEGIN                                         no parameters
    :
    :
    END

PROC EVALUATE (: OUT'P1, OUT'P2);               procedure
    BEGIN                                         no input parameters
    :                                             2 output parameters
    :
    END

PROC PRIME (NUMBER : FACTORTAB) B 1;
    BEGIN                                       function
    :                                             returns Boolean value
    :                                             1 input parameter
    END                                           1 output parameter
```

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'FLOAT F;
ITEM VALUE SINGLE'FLOAT = 3.0;
ITEM ANSWER SINGLE'FLOAT = 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

RECURSION

```
PROC RFACTORIAL REC (IN'ARG) U;
     BEGIN      "PROC"
     ITEM IN'ARG U;
     IF IN'ARG <= 1;
          RFACTORIAL = 1;
     ELSE
          RFACTORIAL = RFACTORIAL (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;
                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                                                  "PROGRAM"
    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 + 1;
    IF (TEST'SUM = NUMBER);                       "TEST IF PERFECT"
        IS'PERFECT = TRUE;
```

SAMPLE PROGRAM 1

```
==================================================================

    PROC FIND'FACTORS (IN'NUMBER, IN'LIMIT : OUT'FACTOR'TAB);
        BEGIN                                    "PROC"
        ITEM IN'NUMBER U;
        ITEM IN'LIMIT U;
        TABLE OUT'FACTOR'TAB (1 : 28);
            ITEM OUT'FACTOR B 1;

        FOR I : 1 BY 1 WHILE I <= IN'LIMIT;
            IF (IN'NUMBER MOD I = 0);
                BEGIN                            "FOUND A FACTOR"
                OUT'FACTOR (I) = TRUE;
                OUT'FACTOR (IN'NUMBER / I) = TRUE;
                END                              "FOUND A FACTOR"
        END                                      "PROC"
    END                                          "PROGRAM"
TERM
```

*-BOUND TABLES - SAMPLE PROGRAM 2

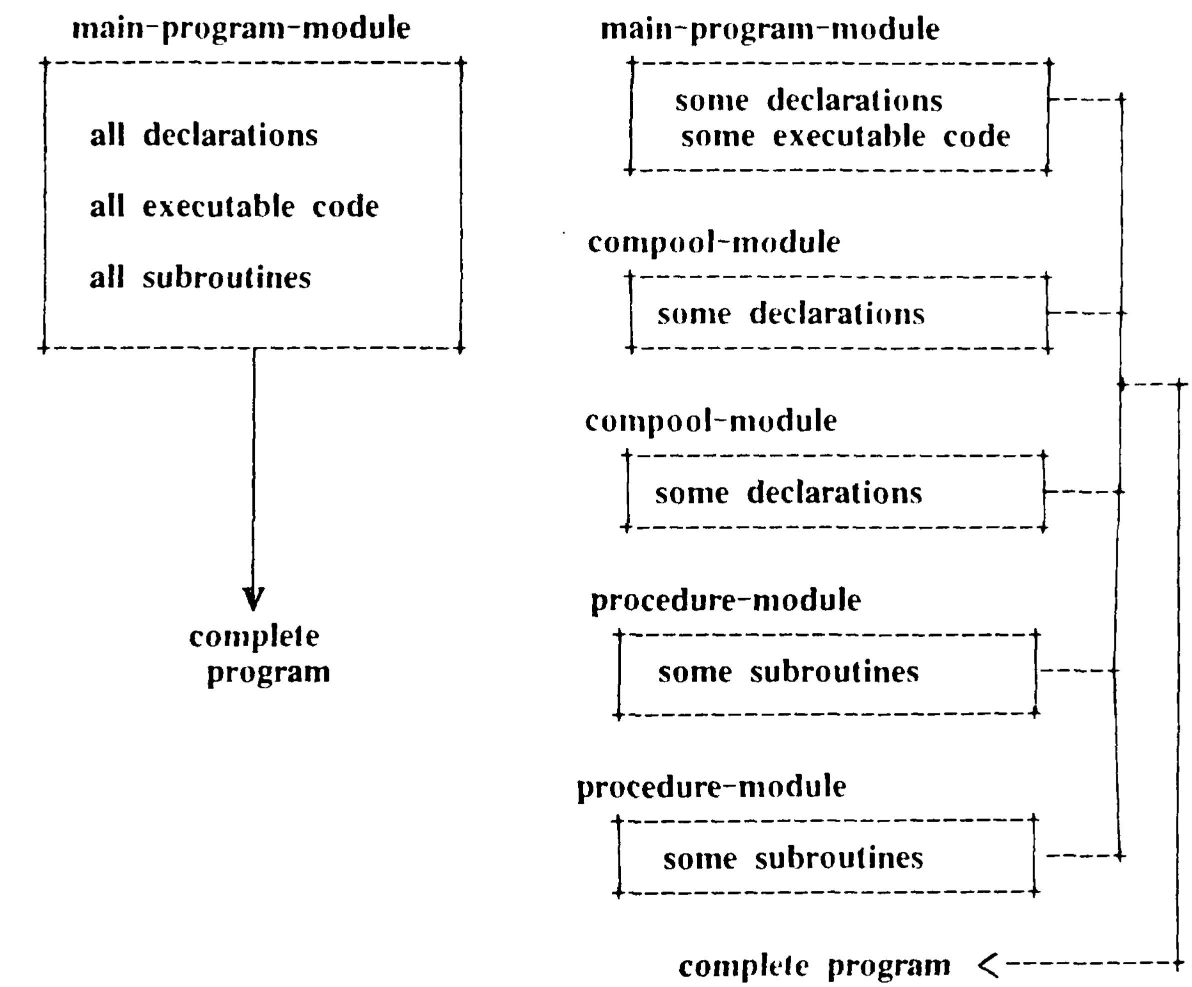
```
===========================================================================

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



DECLARATIONS

- non-executable

- declare a name and attributes associated with that name

- all data names must be declared

DATA STORAGE

STATIC	AUTOMATIC
allocated at beginning of program	allocated when subroutines are invoked
deallocated at end of program	deallocated when subroutines terminate
data objects not declared in subroutines or with STATIC or constants	variables declared in subroutines

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)

SCOPE

```
PROC P1                                  - declared and
+----------------------------------+       useable in P1
|  BEGIN                           |          P1
|  ITEM AA                         |          AA
|  ITEM BB                         |          BB
|  ITEM CC                         |          CC
|     :                            |          P2
|  PROC P2                         |     - declared and
|  +---------------------------+   |       useable in P2
|  |  BEGIN                    |   |          P2
|  |  ITEM ZZ                  |   |          ZZ
|  |  TABLE YY                 |   |          YY
|  |     :                     |   |     - also available in P2
|  |  END                      |   |          P1
|  +---------------------------+   |          AA
|  END                             |          BB
+----------------------------------+          CC
```

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

SCOPE

```
PROC P1
  BEGIN
  ITEM AA S 15;
  ITEM BB
  ITEM CC
     :
  PROC P2
    BEGIN
    ITEM AA F 23;
    ITEM ZZ
    TABLE YY
       :
    END
  END
```

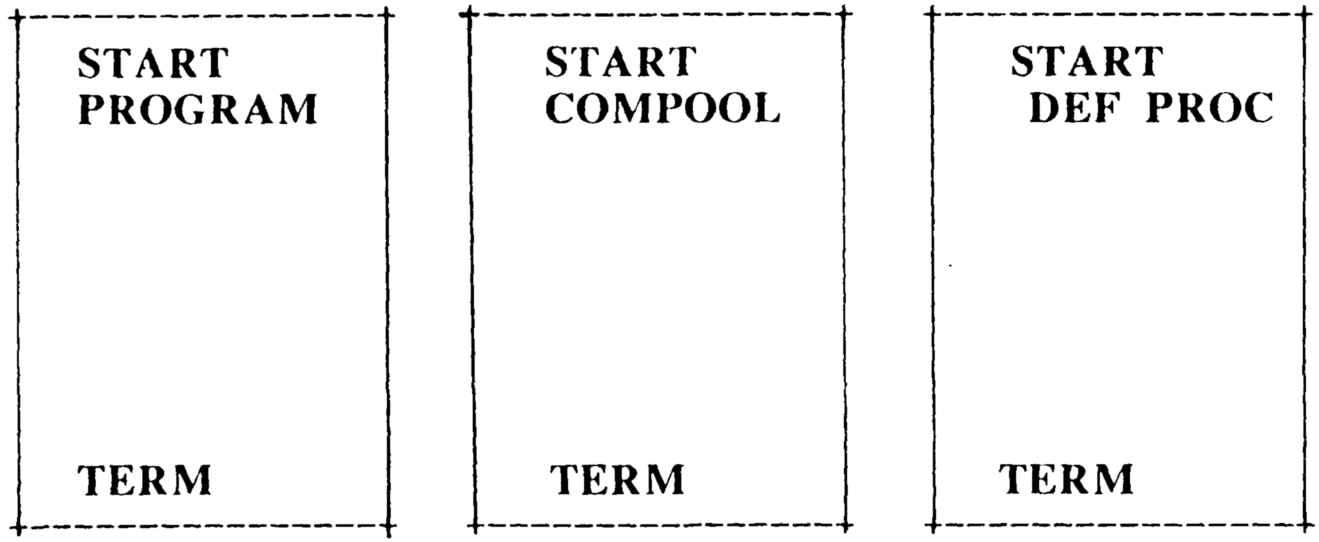
integer AA has effect here (PROC P1, from BEGIN through PROC P2)

floating point AA has effect here; integer AA is not directly accessible (PROC P2, from BEGIN through END)

SCOPE

```
=================================================================
                                              names "visible"
                                              "information
    PROC P1                                    hiding"
    +-------------------------------------------+
    |  BEGIN                                    |
    |  ITEM AA                                  |
    |  ITEM BB                                  |
    |  TABLE ZZ                                 |
    |    :  --------------------------------------|---->  P1,P2,P4
    |  PROC P2                                  |        AA,BB,
    |  +--------------------------------------+ |        TABLE ZZ
    |  |  BEGIN                               | |
    |  |  ITEM HH                             | |
    |  |     :  -----------------------------|---|---> P2,P3,P4
    |  |  PROC P3                             | |        HH
    |  |  +------------------------------+    | |        P1,AA,BB,
    |  |  |  BEGIN                       |    | |        TABLE ZZ
    |  |  |  TABLE II                    |    | |
    |  |  |     :  ----------------------|----|-|----> P3
    |  |  |  END                         |    | |        II
    |  |  +------------------------------+    | |        P2,HH,P4
    |  |  END                                 | |        P1,AA,BB,
    |  +--------------------------------------+ |        TABLE ZZ
    |  PROC P4                                  |
    |  +--------------------------------------+ |
    |  |  BEGIN                               | |
    |  |  ITEM ZZ                             | |
    |  |  ITEM YY                             | |
    |  |     :  -----------------------------|---|---> P4
    |  |  END                                 | |        ITEM ZZ,YY
    |  +--------------------------------------+ |        P2
    |  END                                      |        P1,AA,BB
    +-------------------------------------------+
```

COMPLETE PROGRAM



main-program-module	compool-module	procedure-module
must have one and only one	zero or more	zero or more

START-TERM are module (compilation unit) delimiters

MAIN-PROGRAM-MODULE

```
===========================================================================

START
PROGRAM  name;
     BEGIN

     "DECLARATIONS"        "STATIC ALLOCATION BY DEFAULT

     "EXECUTION"           "EXECUTION BEGINS HERE"

     "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            |      ?      | START            |
| PROGRAM PP;      |   <=====    | PROC P1;         |
|                  |             | PROC P2;         |
|   :              |             |   :              |
| TERM             |             | 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 eslewhere

used on data-names and subroutine-names

EXTERNAL-DECLARATIONS

```
START
PROGRAM SAMPLE;
    BEGIN
    REF PROC SUM (: OUT);  )
        BEGIN              }  ---->  external
        ITEM OUT S;        }         subroutine-declaration
        END                )
    REF ITEM CALLED B 1; }    ---->  external item-declaration
    ITEM VALUE S = 10;

    SUM (: VALUE);       }    ---->  call to external subroutine
    CALLED = TRUE;       }    ---->  reference of external item
    END
TERM
```

EXTERNAL-DECLARATIONS

```
START
DEF ITEM CALLED B 1;          } ----> external item-declaration
DEF PROC SUM (: OUT);         }
      BEGIN                   }
      ITEM OUT S;             } ----> external
      OUT = OUT + 1;          }       subroutine-definition
      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

COMPLETE 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
```

TABLE LAYOUT

ORDINARY TABLE

compiler determines how to position items
in a table

this is the default

DEFAULT

==

```
TABLE DATA (1 : 20);               ordinary
     BEGIN                          serial
     ITEM U3 U 3;                   entry-by-entry
     ITEM B1 B 1;                   1 item / word
     ITEM B3 B 3;
     END

+--------------------------------+
|             U3 (1)             |
+--------------------------------+
|             B1 (1)             |
+--------------------------------+
|             B3 (1)             |
+--------------------------------+
|             U3 (2)             |
+--------------------------------+
|             B1 (2)             |
+--------------------------------+
|             B3 (2)             |
+--------------------------------+
:                                :
:                                :
+--------------------------------+
|             U3 (20)            |
+--------------------------------+
|             B1 (20)            |
+--------------------------------+          20 entries
|             B3 (20)            |           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;        ordinary
     BEGIN                    serial
     ITEM U3 U 3;             entry-by-entry
     ITEM B1 B 1;             dense packing
     ITEM B3 B 3;                3 items / word
     END
```

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

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;     ordinary
      BEGIN                       parallel
      ITEM U3 U 3;                 all first words
      ITEM B1 B 1;                     :
      ITEM B3 B 3;                all last words
      END                         1 item / word
```

```
+------------------------------------------+
|                  U3 (1)                  |
+------------------------------------------+
:                                          :
+------------------------------------------+
|                  U3 (20)                 |
+------------------------------------------+
|                  B1 (1)                  |
+------------------------------------------+
:                                          :
+------------------------------------------+
|                  B1 (20)                 |
+------------------------------------------+
|                  B3 (1)                  |
+------------------------------------------+
:                                          :
+------------------------------------------+    20 entries
|                  B3 (20)                 |     3 words / entry
+------------------------------------------+    60 words
```

STRUCTURE

```
TABLE DATA (1 : 20) T 8;          ordinary
      BEGIN                       serial
      ITEM U3 U 3;                tight structure
      ITEM B1 B 1;                  8 bits-per-entry
      ITEM B3 B 3;                items densely packed by default
      END
```

U3 (1)	B1(1)	B3 (1)	X	U3 (2)	B1(2)	B3(2)	X
U3 (3)	B1(3)	B3 (3)	X	U3 (4)	B1(4)	B3(4)	X
:							:
U3 (20)	B1(20)	B3 (20)	X	U3 (20)	B1(20)	B3(20)	X

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

each item explicitly positioned by the programmer

no packing

items may share storage

fixed-length-entry or variable-length-entry

used to overlay data structures

used to interface with a peripheral

FIXED-LENGTH

```
TABLE INFO (1 : 10) W 3;                              specified
    BEGIN                                             3 words / entry
    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
```

```
bit 0                     8               12      15
   +----------------------+---------------+--------+
   |<------------- NAME (1) --------------------->|
   | INITIAL (1)          |                        |   word 0
   +----------------------+---------------+--------+
   | AGE (1)              | SCHOOL (1)             |   word 1
   +----------------------+---------------+--------+
   |                                      | RANK (1)|  word 2
   +----------------------+---------------+--------+
   :                                               :
   :                                               :
   +----------------------+---------------+--------+
   |<------------- NAME (10) -------------------->|
   | INITIAL (10)         |                        |   word 0
   +----------------------+---------------+--------+
   | AGE (10)             | SCHOOL (10)            |   word 1
   +----------------------+---------------+--------+
   |                                      | RANK (10)| word 2
   +----------------------+---------------+--------+
```

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

OVERLAY-DECLARATION

purposes:

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

OVERLAY-DECLARATION

share storage:

```
OVERLAY AA : BB;
```

specific address:

```
OVERLAY POS (892): CC;
```

specific order:

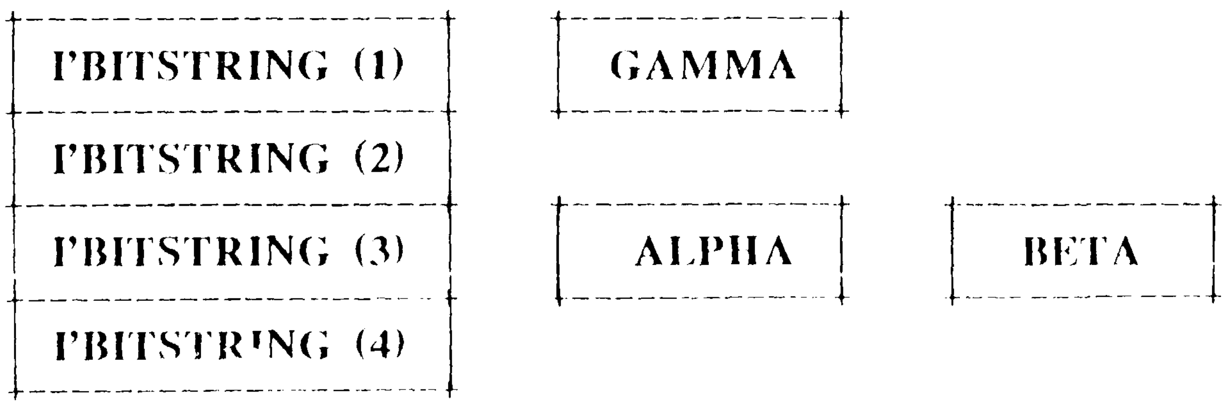
```
OVERLAY DD, EE, FF;
```

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;


+---------------------------+---------------------------+
| <------- I'LONG'FLOAT (1) ---------->                 |
| <------- I'SHORT'FLOAT (1) --------->                 |
+---------------------------+---------------------------+
| I'LONG'FLOAT (1)          |  I'LONG'FLOAT (2)         |
| EXCESS (1)                | I'SHORT'FLOAT (2)         |
+---------------------------+---------------------------+
| <------- I'LONG'FLOAT (2) ---------->                 |
| I'SHORT'FLOAT (2)         |  EXCESS (2)               |
+---------------------------+---------------------------+
:                                                       :
```

OVERLAY-DECLARATION

```
TABLE BIT'STRINGS (1 : 4);
     ITEM I'BITSTRINGS B 16;

ITEM ALPHA U;

ITEM BETA S;

ITEM GAMMA C 2;

OVERLAY BIT'STRINGS : GAMMA, W 1 (ALPHA : BETA);
```



DEFINE

DEFINE

==

- macro
- text substitution
- possibly with parameters

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'I   !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'I   !B";
```

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

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

(second expansion):
                 DEF TABLE MATRIX1 (1 : 3, 1 : 3) = 9 (5.0);
                       ITEM MATRIX1'I SINGLE'FLOAT;
```

BUILT-IN FUNCTIONS

Function	Description	Example
LOC	returns machine-address of its argument (used with dereference)	ITM = VALUE @ LOC (TAB1);
ABS	returns absolute value of its argument	TEN = ABS (-10); FTEN = ABS (5.0 - 15.0);
SGN	returns indicator of sign of its argument (-1 = negative 0 = zero +1 = positive)	NEG'ONE = SGN (-3.2);

BUILT-IN FUNCTIONS

BIT	– select substring of bits	B5 = BIT (B10, 2, 5);
	– pseudo-variable; assign to substring of bits	BIT (B16, 0, 2) = B2;
BYTE	– select substring of characters	C4 = BYTE (C10, 3, 4);
	– pseudo-variable; assign to substring of characters	BYTE (C5, 0, 1) = C1;
REP	– returns machine representation of its argument	B16 = REP (SPEED);
	– pseudo-variable; change machine representation of its argument	REP (C1) = B8;

BUILT-IN FUNCTIONS

SHIFTL	logical left shift of bit string	B110 = SHIFTL (B111);
SHIFTR	logical right shift of bit string	B001 = SHIFTR (B011);
BITSIZE	returns number of bits allocated	SIXTEEN = BITSIZE (S15);
BYTESIZE	returns number of bytes allocated	TWO = BYTESIZE (S15);
WORDSIZE	returns number of words allocated	ONE = WORDSIZE (S15);

BUILT-IN FUNCTIONS

Function	Description	Example
NWDSEN	returns number of words in a table-entry	`TABLE TAB1; BEGIN ITEM CC C 4; ITEM UU U; END THREE = NWDSEN (TAB1);`
FIRST	returns highest-valued status-constant	`TYPE LETTER STATUS ( V(A), V(B), V(C) ); AA = FIRST (LETTER);`
LAST	returns lowest-valued status-constant	`TYPE LETTER STATUS ( V(A), V(B), V(C) ); CC = LAST (LETTER);`

BUILT-IN FUNCTIONS

Function	Description	Example
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

AD-A142 780 PROCEEDINGS PAPERS OF THE AFSC (AIR FORCE SYSTEMS COMMAND) AVIONICS STAND..(U) AERONAUTICAL SYSTEMS DIV WRIGHT-PATTERSON AFB OH DIRECTORATE O.. 3/3

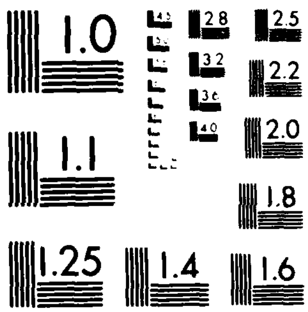
UNCLASSIFIED C A PORUBCANSKY NOV 82 F/G 9/2 NL

END

DATE FILMED

8 84

DTIC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0

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

ED

84