

AD-A165 352

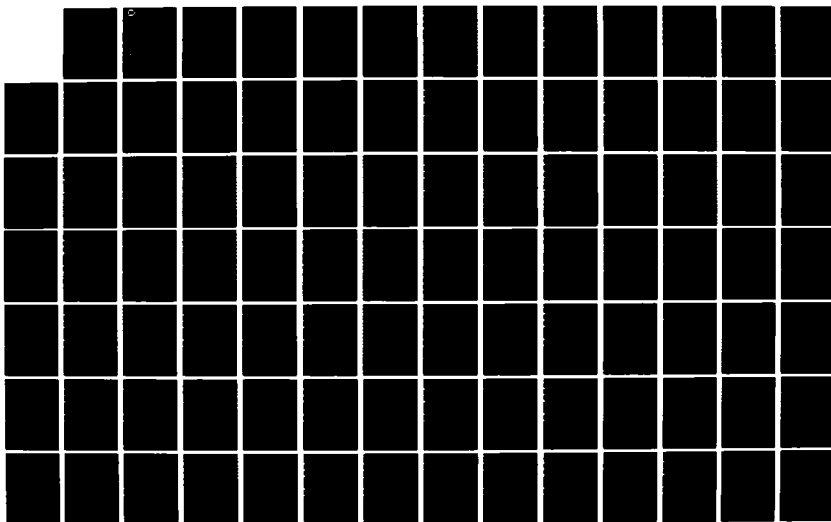
ADA (TRADEMARK) TRAINING CURRICULUM: ADA TECHNICAL
OVERVIEW L102 TEACHER'S GUIDE(U) SOFTECH INC WALTHAM MA
1986 DAAB77-83-C-K506

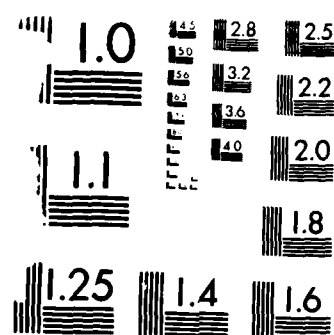
1/3

UNCLASSIFIED

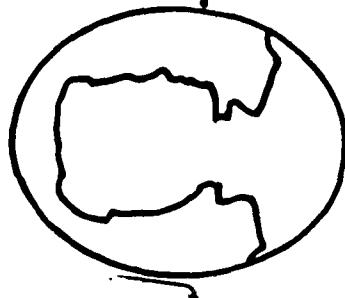
F/G 5/9

NL



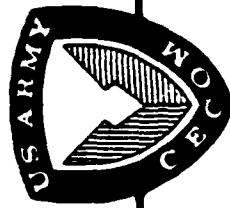


MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A



Ada® Training Curriculum

1986



DTIC

AD-A165 352

Ada® Technical Overview L102 Teacher's Guide

86 3 11 14

Submitted 1/4/86

D

MAR 12 1986

Prepared By:

SOFTECH, INC.

460 Totten Pond Road
Waltham, MA 02154

U.S. Army Communications-Electronics Command
(CECOM)

Contract DAAB07-83-C-K506

INSTRUCTOR NOTES

THE OBJECTIVE OF THIS MODULE IS TO PROVIDE AN INTRODUCTION TO THE ADA LANGUAGE. THE STUDENT SHOULD GAIN A BEGINNER'S READING KNOWLEDGE OF ADA AND A GOOD FOUNDATION FOR CONTINUED LEARNING. NOTE: THIS MODULE DOES NOT TEACH ADA, BUT TEACHES ABOUT ADA.

BRIEFLY GIVE AN OVERVIEW OF WHAT WILL BE COVERED IN THE MODULE. THE APPROACH IS LEARNING ABOUT ADA THROUGH ADA EXAMPLES. SYNTAX IS NOT STRESSED OR EVEN COVERED. THIS MATERIAL IS FOR OTHER MODULES OF THE CURRICULUM. TELL THE STUDENTS THAT LEARNING ADA IS AN ITERATIVE PROCESS: THEY LEARN SOME, USE IT, AND LEARN SOME MORE. THUS IT IS NOT IMPERATIVE THAT THEY GRASP ALL THE FINE DETAILS. THEY SHOULD AIM FOR THE CONCEPTS AND INTUITIVE "FEEL" OF THE LANGUAGE.

Copyright by SofTech, Inc. 1984. This material may be reproduced by or for the U.S. Government pursuant to the copyright license under DAR clause 7-104.9(a) (May 81).

1. Mission For ☒ ☐ ☐	2. Date 1944	3. Place Washington	4. Description Availability Codes Available for Special	5. Remarks A-1
---	-----------------	------------------------	--	-------------------

INSTRUCTOR NOTES

THIS SECTION SETS THE HISTORICAL MOTIVATION FOR DoD AND THE RESULTING ADA EFFORT. IT ALSO OUTLINES ITS DEVELOPMENT HISTORY.

ALLOW 60 MINUTES FOR THIS SECTION.

Section 1

Background and Rationale for Ada

INSTRUCTOR NOTES

TOPIC OUTLINE

BACKGROUND AND RATIONALE FOR ADA

WRITING AN ADA PROGRAM FROM BEGIN TO END

SUMMARY OF ADA PROGRAM STRUCTURE

ADA THROUGH EXAMPLE

LARGE SYSTEM DEVELOPMENT

SUMMARY OF ADA FEATURES

FOR MORE INFORMATION

INSTRUCTOR NOTES

A LIST THAT CHARACTERIZES THE PRESENT STATE OF SOFTWARE DEVELOPED FOR EMBEDDED COMPUTER SYSTEMS.

SOFTWARE CRISIS: MOTIVATION FOR ADA

SOFTWARE FOR COMPLEX MILITARY SYSTEMS

- IS USUALLY LATE
- COSTS MORE THAN ORIGINALLY ESTIMATED
- DOES NOT WORK TO ORIGINAL SPECIFICATIONS
- IS UNRELIABLE
- IS DIFFICULT AND COSTLY TO MAINTAIN

INSTRUCTOR NOTES

FOLLOWING ARE SEVERAL GRAPHS AND A LIST OF UNDERLYING PROBLEMS ASSOCIATED WITH THIS

"SOFTWARE CRISIS"

BRIEFLY GO THROUGH THESE.

PROBLEMS ASSOCIATED WITH THE SOFTWARE CRISIS

VG 732.1

1-3

INSTRUCTOR NOTES

IN 1965, COST OF DEVELOPING A SOFTWARE SYSTEM WAS PRIMARILY A HARDWARE COST.

AROUND 1970 THIS BREAKDOWN OF TOTAL COST OF A SYSTEM WAS SPLIT FAIRLY EVENLY BETWEEN
HARDWARE AND SOFTWARE.

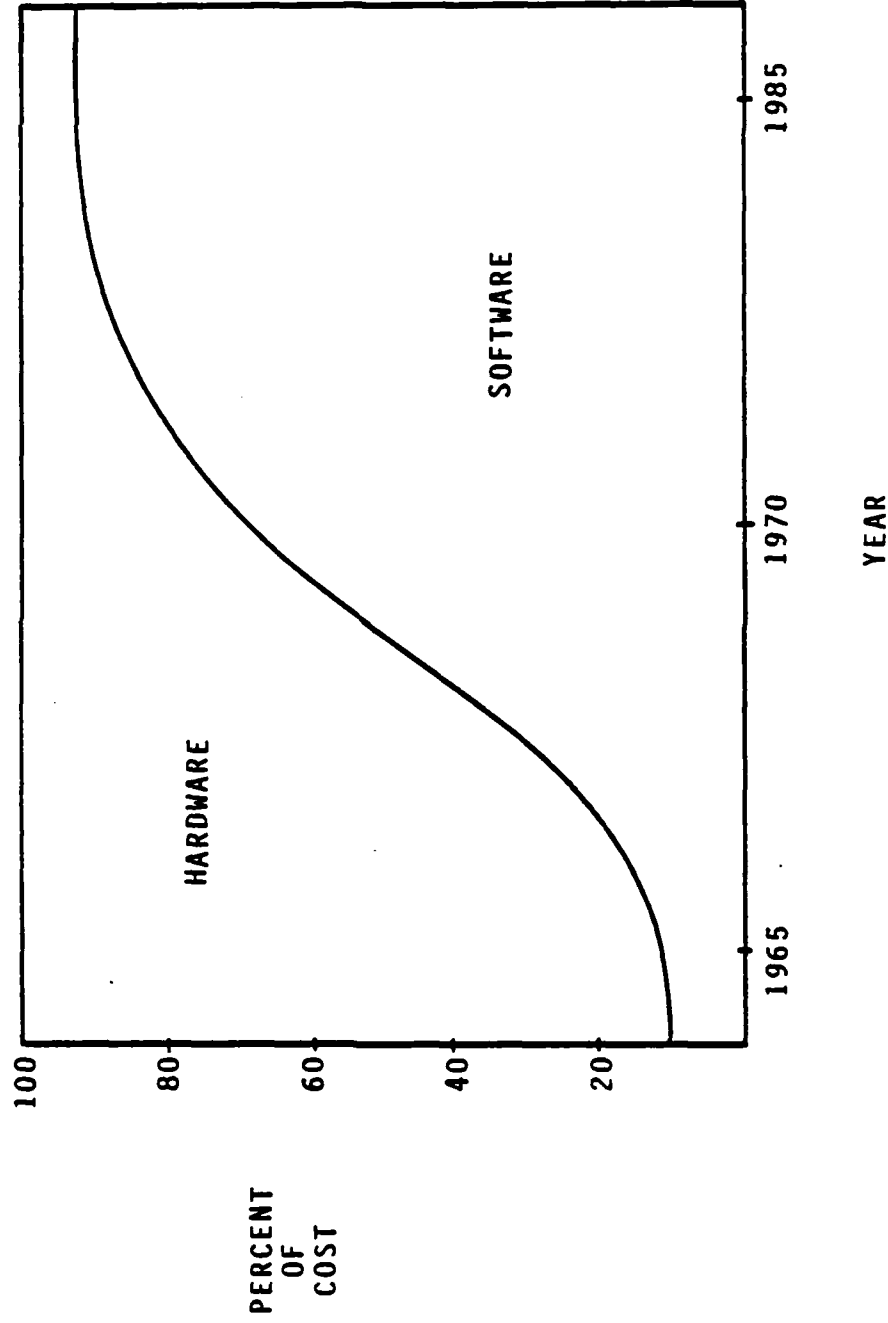
BUT SINCE THEN, SOFTWARE COSTS FOR A SYSTEM HAVE RISEN DRAMATICALLY WHILE HARDWARE COSTS
HAVE PLUMMETED AS A RESULT OF MICRO-CHIP TECHNOLOGICAL ADVANCES.

SOURCE: BARRY BOEHM, DEC 1976, IEEE TRANSACTIONS.

VG 732.1

1-4i

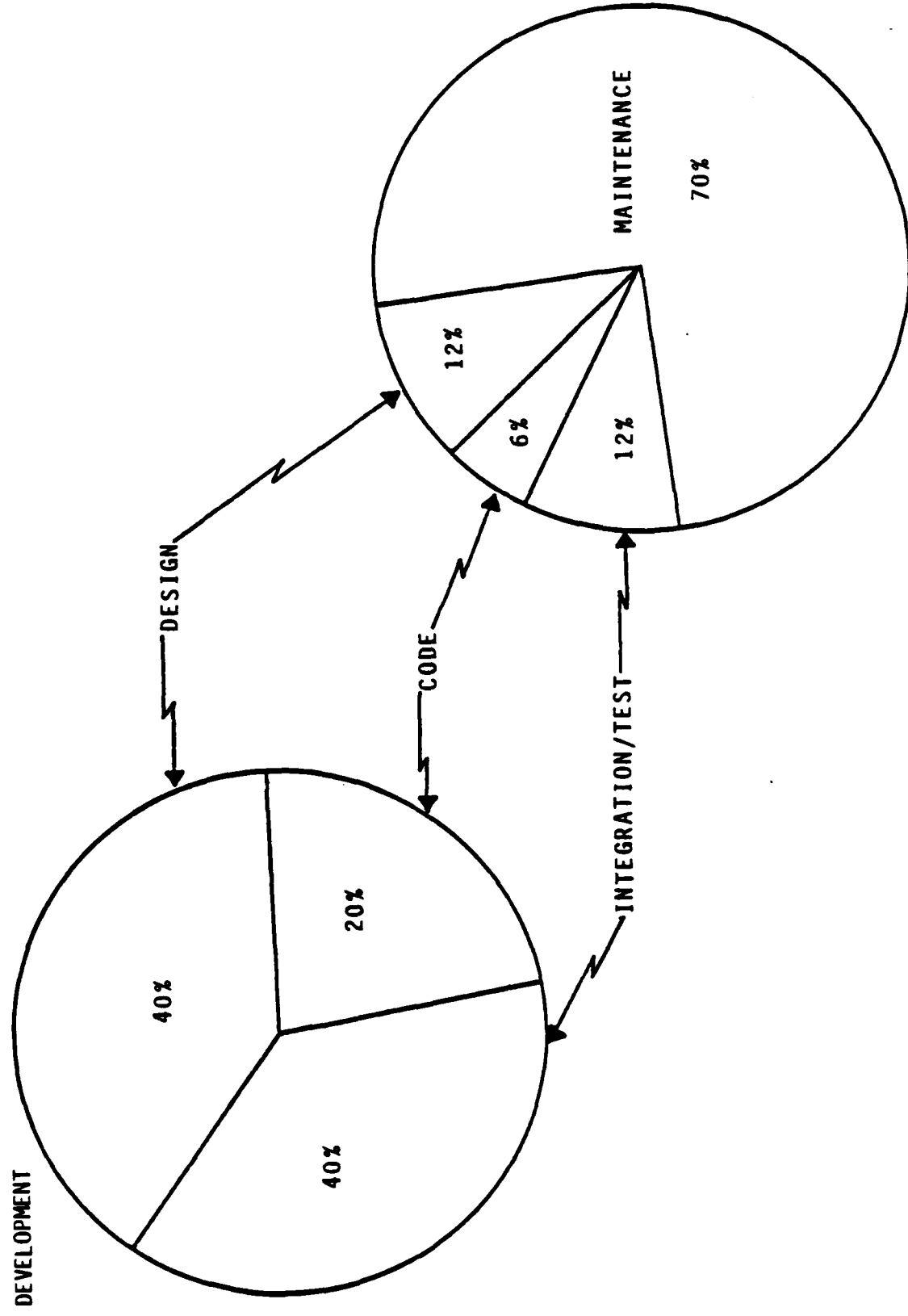
SOFTWARE COSTS INCREASING AS HARDWARE COSTS DECREASING



INSTRUCTOR NOTES

THE CAUSE OF THE INCREASED SOFTWARE COSTS IS THE SPECIFIC COST OF MAINTAINING/UPGRADING
A SYSTEM ONCE IT IS OPERATIONAL.

SOFTWARE MAINTENANCE NEARLY TRIPLE ORIGINAL DEVELOPMENT COSTS



INSTRUCTOR NOTES

AN ADDITIONAL COST WITH SOFTWARE LIES IN ERROR DETECTION AND CORRECTION.

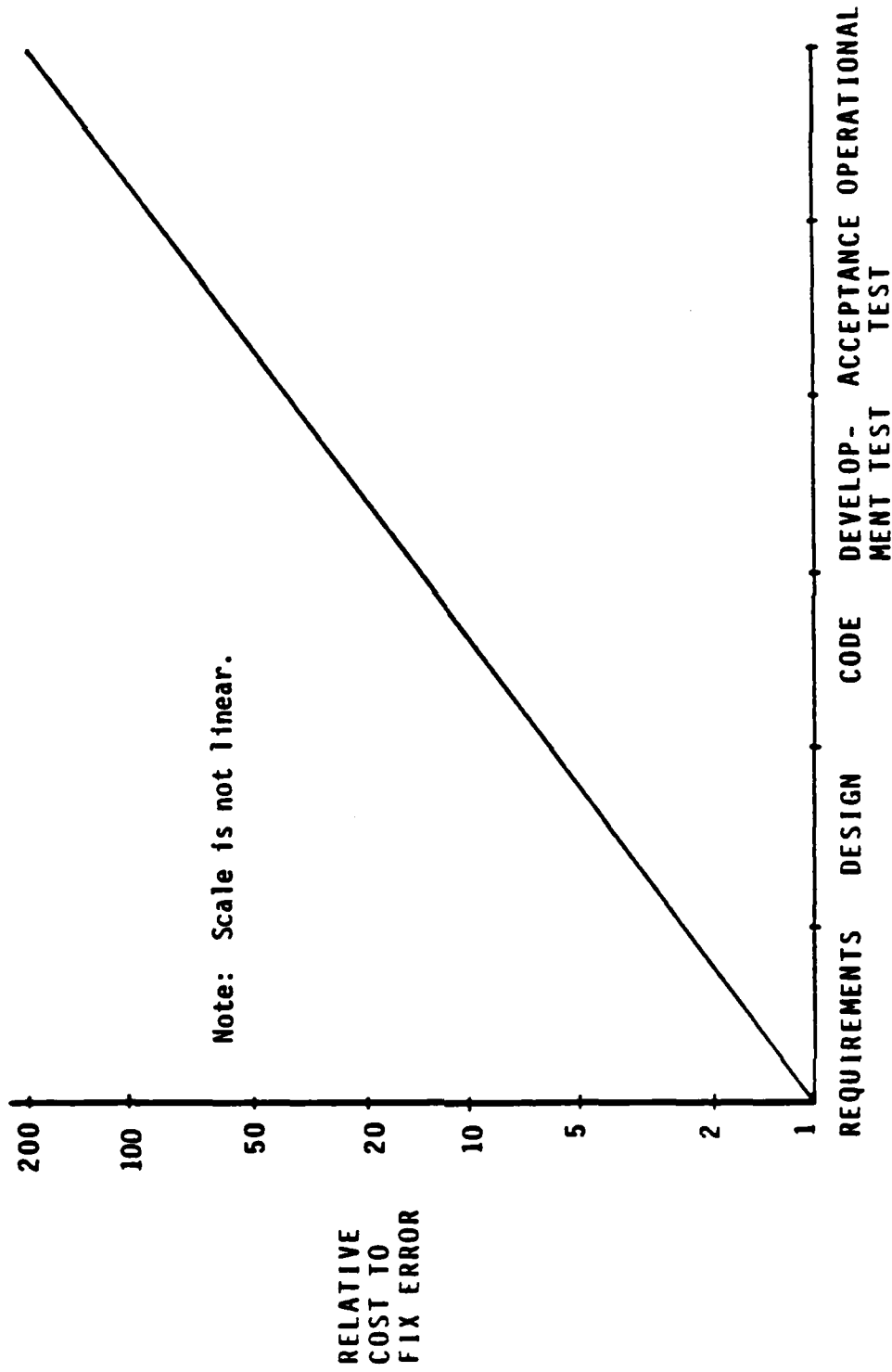
FOR EXAMPLE:

IF A REQUIREMENTS ERROR IS FOUND AND CORRECTED DURING THE REQUIREMENTS PHASE, YOU CAN JUST CORRECT THE REQUIREMENTS DOCUMENT WITH LITTLE COST IMPACT OF THE ERROR.

IF THE SAME ERROR IS NOT FOUND AND CORRECTED UNTIL MAINTENANCE, THE CORRECTION INVOLVES NOT ONLY DOCUMENT CHANGES SUCH AS SPECIFICATIONS, USER MANUALS, TRAINING MANUALS, BUT WILL ALSO INVOLVE VARIOUS AMOUNTS OF CODE MODIFICATIONS AND REVALIDATION. ERROR CORRECTION AT THIS POINT IN THE LIFE CYCLE IS TYPICALLY 100 TIMES WHAT IT WOULD HAVE BEEN IN THE REQUIREMENTS PHASE. THUS UPDATING DOCUMENTATION BECOMES A MAJOR COST FACTOR.

SOURCE: B. BOEHM, SOFTWARE ENGINEERING ECONOMICS, 1981
DATA IS FROM STUDIES BY IBM, TRW, GTE ON THIS TOPIC

COST OF ERROR CORRECTION



PHASE ERROR DETECTED AND CORRECTED

INSTRUCTOR NOTES

OTHER ASSOCIATED PROBLEMS WITH DECREASED PRODUCTIVITY AND RELIABILITY OF OUR SOFTWARE ARE THAT THE PROBLEMS WE ARE ATTEMPTING TO SOLVE NOW ARE MUCH MORE COMPLEX THAN IN THE PAST. COMPLEXITY ALONE IS NOT A PROBLEM, IT'S THE LACK OF ADEQUATE TOOLS TO ASSIST.

ADDITIONAL PROBLEMS

- SOFTWARE TASKS ARE MORE COMPLEX NOW, BUT NO ADEQUATE TOOLS TO DEAL WITH THE PROBLEM
- SUPPORT TOOLS (ASSEMBLERS, LINKERS, DEBUGGER) MUST BE DEVELOPED FOR EACH LANGUAGE AND MACHINE
- LACK OF ADEQUATE MANAGEMENT AND SOFTWARE DEVELOPMENT TOOLS

INSTRUCTOR NOTES

AS ARCHITECTURES HAVE PROLIFERATED, SO TOO HAVE LANGUAGES. PLUS THE SUPPORT TOOLS FOR EACH ARCHITECTURE/LANGUAGE COMBINATION MUST BE DEVELOPED ANEW. OUR CURRENT LANGUAGES ARE NOT WELL SUITED TO THE NEEDS OF EMBEDDED COMPUTER SYSTEMS.

ADDITIONAL PROBLEMS (Continued)

- SOFTWARE IS NOT REUSABLE ON DIFFERENT SYSTEMS
- PROLIFERATION OF LANGUAGES AND ARCHITECTURES
- LANGUAGES NOT SUITED FOR CURRENT APPLICATION
- SUPPLY OF QUALITY SOFTWARE PERSONNEL NOT ABLE TO MEET CURRENT SOFTWARE DEMANDS

INSTRUCTOR NOTES

IT IS A RETHINKING OF THE WAY IN WHICH SOFTWARE SYSTEMS WILL BE DEVELOPED IN THE FUTURE WITH THE ITEMS LISTED AS VEHICLES OF THAT CHANGE. NOTE THAT IT IS THE COMBINATION OF LANGUAGE, ENVIRONMENT, AND METHODOLOGIES THAT CONSTITUTES THE ADA EFFORT.

WHEN WE SPEAK OF MODERN SOFTWARE ENGINEERING METHODS, WE ARE REFERRING TO SUCH THINGS AS STRUCTURED DESIGN AND PROGRAMMING, TOP-DOWN DEVELOPMENT, STRONG DATA TYPING, MODULARITY.

RELIABLE SOFTWARE IMPLIES THAT THE SOFTWARE PRODUCT CAN RECOVER FROM ERROR OR FAILURE CONDITIONS IN OPERATION AS WELL AS PREVENT ERRORS IN ANALYSIS, DESIGN, AND CODE IMPLEMENTATION.

MAINTAINABLE SOFTWARE IMPLIES THAT OUR SOFTWARE PRODUCT HAS BEEN CONSTRUCTED SUCH THAT THE STRUCTURE AND ORGANIZATION OF THE SYSTEM ARE CLEAR AND MODIFICATION TO THE SYSTEM CAN BE DONE WITH RELATIVE EASE (SUCH THAT CHANGES DO NOT CAUSE NEW ERRORS).

COST REDUCTION OCCURS ONLY OVER THE LIFE OF THE PRODUCT. WE ARE PRIMARILY CONCERNED WITH PROJECTS OF LONG DURATION WHICH WILL BE MODIFIED AND ENHANCED CONTINUALLY. THERE IS NO COST SAVINGS DURING DEVELOPMENT.

THE ADA EFFORT: DoD's RESPONSE

THROUGH A COMBINATION OF:

- MODERN SOFTWARE ENGINEERING METHODS
- COMMON HIGH ORDER LANGUAGE (ADA)
- COMMON SUPPORT TOOLS (ADA PROGRAMMING SUPPORT ENVIRONMENT - APSE)

DEVELOP SOFTWARE THAT IS:

- RELIABLE
- MAINTAINABLE
- LESS COSTLY OVER THE LIFE CYCLE
- PORTABLE

INSTRUCTOR NOTES

THE APPROACH TO THE ADA DESIGN WAS INNOVATIVE. A LIFE-CYCLE APPROACH WAS TAKEN. THE ADA LANGUAGE CAN BE VIEWED AS A PRODUCT LIKE BUILDING A MISSILE: FROM ANALYSIS OF A PROBLEM AND POSSIBLE SOLUTION, THROUGH REQUIREMENTS (IN THE SERIES OF LANGUAGE REQUIREMENT SPECS), TO OPERATIONAL (WITH ACTUAL COMPILER DEVELOPMENT AND VALIDATION).

IMPORTANT TO NOTE THAT THROUGHOUT THE PROCESS, UNIVERSITIES, INDUSTRY AND COMPILER IMPLEMENTORS WERE SOLICITED FOR INPUT (REVIEWS, OPINIONS).

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000

DEVELOPMENT OF ADA LANGUAGE

ANALYSIS	1970-1975	IDENTIFICATION OF SOFTWARE PROBLEMS IN EMBEDDED MILITARY SYSTEMS (THE CRISIS) STRAWMAN, WOODENMAN, TINMAN LANGUAGE REQUIREMENTS SPECIFICATIONS
REQUIREMENTS	1975-1977	HOLWG: HOL REQUIREMENTS FOR EMBEDDED SYSTEMS DEFINED EXISTING LANGUAGES EVALUATED RESULTS: ONE LANGUAGE IS SUFFICIENT NO EXISTING LANGUAGE SATISFIES ALL REQUIREMENTS AN EXISTING LANGUAGE SHOULD BE USED AS A BASE
DESIGN		
PHASE I	1977-1978	PRELIMINARY LANGUAGE DESIGN - IRONMAN (RED, BLUE, YELLOW, GREEN)
PHASE II	1978-1979	FORMAL LANGUAGE DEFINITION - STEELMAN (RED, GREEN)
PHASE III	1979-1980	FINAL LANGUAGE DEFINITION BY CII HONEYWELL/BULL

INSTRUCTOR NOTES

COMPILER VALIDATION INSTITUTED TO RESTRICT THE PROLIFERATION OF ADA DIALECTS. ADA COMPILERS MUST BE VALIDATED YEARLY AND IF A NEW VERSION IS RELEASED BY THE ADA VALIDATION OFFICE (PART OF THE ADA JOINT PROGRAM OFFICE-AJPO).

PARALLEL PROJECTS ALLOW FOR AN ORGANIZATION TO TRANSITION METHODICALLY TO ADA BY DOING A PARTICULAR PROJECT IN ADA AND IN THE FORMER LANGUAGE AND METHODS. THUS EXPERIENCE INTO ADA METHODS CAN BE EXPLORED WITHOUT IMPACT TO THE END PRODUCT.

DR. DELAUER'S PROCLAMATION MANDATES THE USE OF ADA ON ALL NEW CONTRACTS AS OF 1 JAN 84.

THE TOTAL NUMBER OF VALIDATED COMPILERS COVERS 11 VENDORS AND MANY COMBINATIONS OF HOST AND TARGET COMPUTERS. THERE ARE 4 VAX 11/750 SYSTEMS AND 7 VAX 11/780, 782, 785 SYSTEMS, TO NAME THE MOST COMMON COMPUTER.

VG 732.1

1-11i

LANGUAGE DEVELOPMENT (Continued)

TESTING	1980-1982	LANGUAGE REFINEMENT BY INTERNATIONAL REVIEWERS COMPILER VALIDATION TEST FACILITY ANSI STANDARDIZATION REQUESTED
OPERATIONAL	1982 	COMPILER DEVELOPMENT BY DoD, PRIVATE INDUSTRY, ACADEMIA
		PARALLEL PROJECTS
	FEB. 1983	ANSI STANDARDIZATION OF ADA LANGUAGE
	MAR. 1983	NYU (ADA/ED) VALIDATED TRANSLATOR
	JUN. 1983	ROLM VALIDATED COMPILER DR. DELAUER'S PROCLAMATION
	DEC. 1984	SOFTTECH ALS VALIDATED
	OCT. 1985	35 VALIDATED COMPILERS

INSTRUCTOR NOTES

WHAT DO WE MEAN BY ENVIRONMENTS IN GENERAL.

ENVIRONMENTS

- PROVIDE A SET OF AUTOMATED TOOLS TO AID SOFTWARE DEVELOPERS AT VARIOUS PHASES IN THE LIFE CYCLE

EXAMPLES:

COMPILERS

LINKERS

LOADERS

CODE AUDITORS

PROGRAMMING SUPPORT LIBRARIES

- CURRENT SITUATION WITH ENVIRONMENTS
MUST BE DEVELOPED FOR EACH MACHINE
PERSONNEL MUST LEARN A NEW SET OF TOOLS FOR EACH MACHINE
LIMITED TOOL SETS AVAILABLE

INSTRUCTOR NOTES

SPECIFICALLY ADA ENVIRONMENTS.

THE APSE WAS INTENDED TO BE HOSTED ON ONE PHYSICAL MACHINE (GENERALLY A SIZABLE MAINFRAME) WITH THE TARGET MACHINE OF THE DEVELOPMENT PROBABLY A MUCH SMALLER COMPUTER (WHICH WOULD NOT HAVE THE ADDRESS SPACE/PERIPHERALS NECESSARY).

THE DATABASE OF THE APSE IS AN IMPORTANT FEATURE. IT HOUSES ALL PROJECT SOURCE CODE, OBJECT CODE, AND DOCUMENTATION.

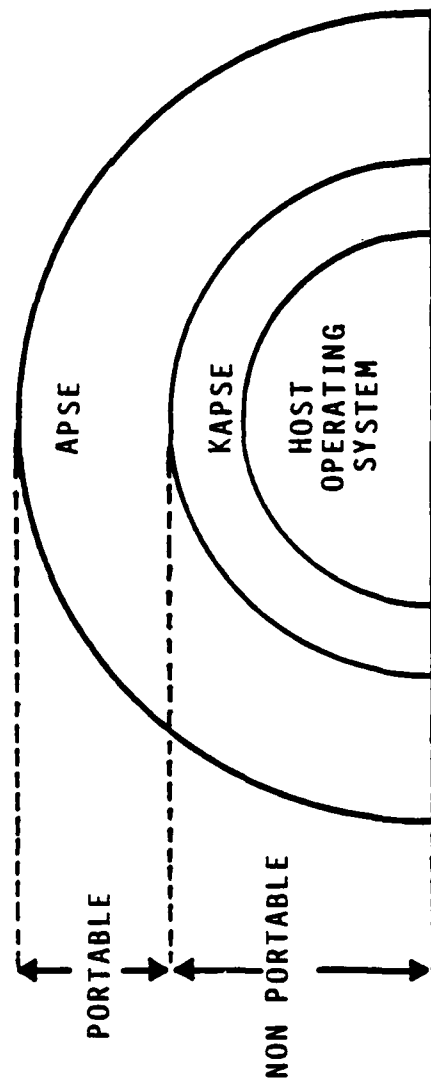
ADA ENVIRONMENTS

- GOAL IS TO PROVIDE AUTOMATED TOOL SUPPORT FOR ALL PROJECT PERSONNEL INVOLVED IN MANAGING, DEVELOPING, AND MAINTAINING SOFTWARE SYSTEMS
- INCLUDES TOOLS FOR ALL PHASES OF LIFE CYCLE
- ADVANTAGES
 - TOOL DEVELOPMENT COSTS REDUCED
 - PORTABILITY OF TOOLS, SOFTWARE, PROGRAMMERS
 - CAN BE USED THROUGHOUT THE LIFE CYCLE
- PORTABILITY ACHIEVED THROUGH A LOW-LEVEL INTERFACE TO THE HOST OPERATING SYSTEM (THE KAPSE) AND A SET OF TOOLS (THE APSE)

INSTRUCTOR NOTES

CONCEPTUALLY THE STRUCTURE IS IN NESTED LEVELS. AT THE INNER MOST LEVEL IN THE OPERATING SYSTEM IS THE PHYSICAL DATABASE. ABOVE IT, IS THE KAPSE WHICH TAKES CARE OF ALL PHYSICAL TO LOGICAL INTERFACES OF THE ENTIRE APSE. ABOVE THE KAPSE, THE APSE SITS. IT CONTAINS TOOLS NECESSARY TO AID SOFTWARE DEVELOPMENT THROUGHOUT THE LIFE CYCLE.

ADA ENVIRONMENT STRUCTURE



KAPSE: KERNEL ADA PROGRAMMING SUPPORT ENVIRONMENT

APSE: ADA PROGRAMMING SUPPORT ENVIRONMENT

INSTRUCTOR NOTES

WHAT IS IN EACH PART OF APSE:

KAPSE: NO EXPLICIT TOOLS BUT SUPPORTS -
 DATABASE ACCESS
 I/O
 TERMINAL TO TOOL ACCESS
 RUNTIME SYSTEM

APSE: TOOLS INCLUDE:

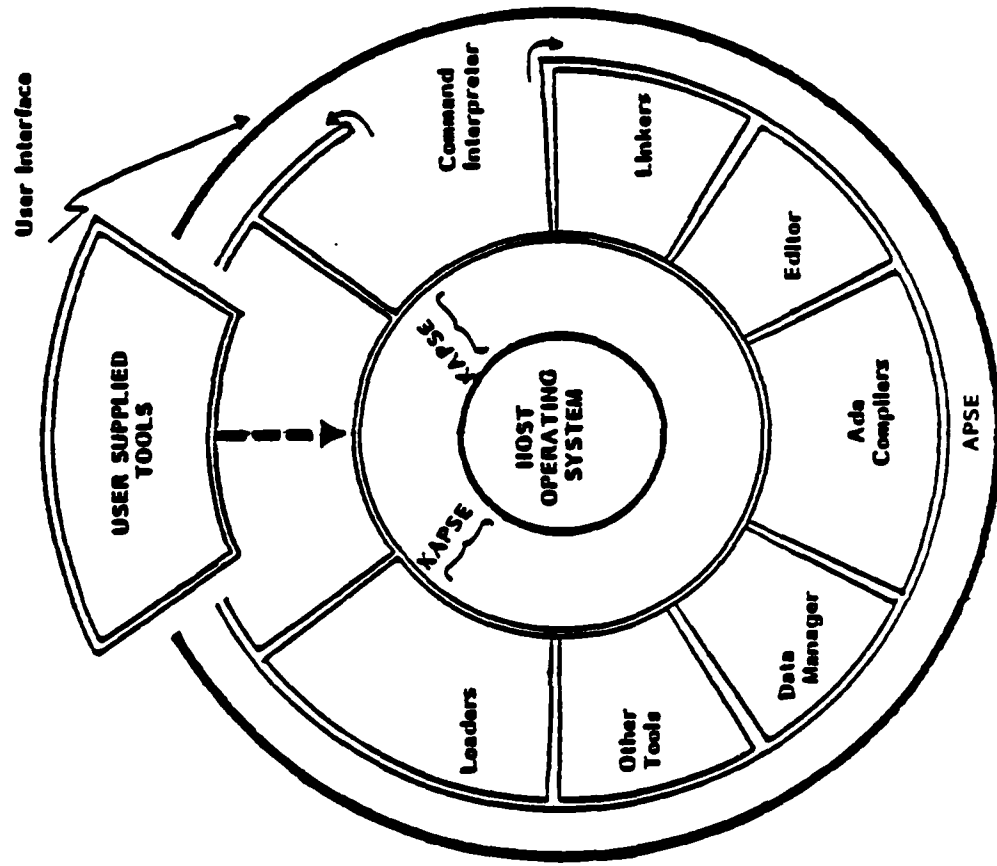
COMPILERS	SYMBOLIC DEBUGGERS
LOADERS	COMMAND INTERPRETER
LINKERS	FILE ADMINISTRATOR TOOLS
TEXT EDITOR	CONFIGURATION MANAGEMENT TOOLS

THE KAPSE SHOULD CONTAIN ALL LOW-LEVEL FEATURES NECESSARY TO REHOST ONTO ANOTHER SYSTEM.

INSTRUCTOR NOTES

THIS IS THE COMMON PICTURE OF THE APSE STRUCTURE THAT THE STUDENT WILL SEE.

APSE STRUCTURE



INSTRUCTOR NOTES

SIMILAR FORMAT AS THE LANGUAGE.

OF NOTE: THE SPECIFICATION FOR THE ENVIRONMENTS IS NOT AS RIGOROUS AS FOR THE LANGUAGE
SINCE WE KNOW LESS OF WHAT SHOULD BE IN AN ENVIRONMENT.

MAIN ENVIRONMENT PROJECTS : ALS (ADA LANGUAGE SYSTEM)

AIE (ADA INTEGRATED ENVIRONMENT)

- IN 1985, AIE WAS DOWNGRADED TO ACS (ADA COMPILATION SYSTEM)

DEVELOPMENT OF ADA ENVIRONMENTS

ANALYSIS	1977-1978	LANGUAGE ALONE NOT SUFFICIENT TO IMPROVE SOFTWARE DEVELOPMENT
REQUIREMENTS	1978-1979	PRELIMINARY ENVIRONMENT REQUIREMENTS (SANDMAN, PEBBLEMAN)
DESIGN	1980	FORMAL ENVIRONMENT DEFINITION (STONEMAN)
IMPLEMENTATION	1981 →	COMPILER PLUS ENVIRONMENT DEVELOPMENT PROJECTS FUNDED BY DoD, PRIVATE INDUSTRY, UNIVERSITIES
TESTING	1982 →	KAPSE INTERFACE TEAM (KIT) FOR INDUSTRY AND ACADEMIA (KITIA): TASK IS TO DEFINE STANDARD INTERFACES FOR ALS AND AIE
OPERATIONAL/ MAINTENANCE	1983 →	

INSTRUCTOR NOTES

THIS RELATES THE ADA EFFORT TO OUR ORIGINAL PROBLEM. HOW OR WHY EACH PART OF THE EFFORT IS USEFUL IN ATTEMPTING TO MANAGE OUR SOFTWARE PROBLEMS. IN THIS PERSPECTIVE, ADA IS NOT JUST A LANGUAGE, BUT BECOMES A TOOL - LIKE LINKERS, DEBUGGERS, METHODOLOGIES - TO DEAL WITH SOFTWARE DEVELOPMENT PROBLEMS.

RELIABILITY AND MAINTAINABILITY ARE INCREASED THROUGH MODERN SOFTWARE ENGINEERING PRINCIPLES AND METHODS SUCH AS STRUCTURED DESIGN AND PROGRAMMING (WHICH ALSO HELP INCREASE PRODUCTIVITY), MODULARITY, STRONG TYPING AND ERROR RECOVERY MECHANISMS.

THE ADA EFFORT AND THE SOFTWARE CRISIS

- MODERN SOFTWARE ENGINEERING METHODS

INCREASED PRODUCTIVITY

INCREASED RELIABILITY, MAINTAINABILITY

- COMMON HIGH ORDER LANGUAGE

DESIGNED TO SUPPORT MODERN SOFTWARE DEVELOPMENT METHODS

SUPPORTS THE MANAGEMENT OF COMPLEXITY AND CHANGING REQUIREMENTS

REDUCED PROGRAMMER RETRAINING

- COMMON SUPPORT ENVIRONMENT

REDUCED COST OF WRITING CUSTOMIZED SYSTEMS PROGRAMS

INCREASED PORTABILITY OF SOFTWARE/PROGRAMMERS

LIFE CYCLE SUPPORT OF SOFTWARE DEVELOPMENT

REDUCED PROGRAMMER RETRAINING

INSTRUCTOR NOTES

THIS SECTION PROVIDES AN OVERVIEW (CONCEPTUAL, INTUITIVE FEEL) OF PROGRAMMING IN ADA,
FROM PROBLEM DEFINITION TO MAINTENANCE.

STRESS TO THE STUDENTS THAT SYNTAX IS NOT THE KEY ISSUE HERE -- OVERALL STRUCTURE AND
CONCEPTS IS.

ALLOW 1-1/2 HOURS FOR THIS SECTION. BREAK BEFORE "COMPILATION."

Section 2

Writing an Ada Program from Begin to End

INSTRUCTOR NOTES

TOPIC OUTLINE

BACKGROUND AND RATIONALE FOR ADA

WRITING AN ADA PROGRAM FROM BEGIN TO END

SUMMARY OF ADA PROGRAM STRUCTURE

ADA THROUGH EXAMPLE

LARGE SYSTEM DEVELOPMENT

SUMMARY OF ADA FEATURES

FOR MORE INFORMATION

INSTRUCTOR NOTES

THE PURPOSE OF THE EXAMPLE IS TO ILLUSTRATE WHAT IT'S LIKE TO WRITE AN ADA PROGRAM FROM BEGINNING TO END. THIS GIVES AN APPRECIATION OF THE PROCESS IN ADA. THIS EXAMPLE IS ELEMENTARY BUT BECAUSE OF THAT, THE STUDENT CAN CONCENTRATE ON THE ADA AND NOT THE ALGORITHMS. THE FORMAT IS TO PARALLEL SOFTWARE DEVELOPMENT. FIRST DECOMPOSE THE PROBLEM FROM THE TOP, DOWN THROUGH SPECIFIC ALGORITHMS TO THE CONTROL STRUCTURE LEVEL. AFTER THUS ANALYZING THE PROBLEM, THE ADA CODE IS BUILT FROM THIS POINT BACK UP TO A COMPLETE ADA SYSTEM. THE ADA SYNTAX IS TOTALLY BY EXAMPLE (I.E. OSMOSIS). ADDITIONAL GOALS ARE TO GENERATE A FAMILIARITY WITH ADA, THE EASE WITH WHICH IT CAN BE READ, AND TO CREATE A NON-THREATENING APPRECIATION FOR THE LANGUAGE. TO BUILD THE ADA SYSTEM, WE START FIRST WITH CONTROL STRUCTURES, AS ACTION STATEMENTS IN ADA ARE VERY SIMILAR TO OTHER LANGUAGES. THE STATEMENT CODE FRAGMENTS ARE SIMILAR TO WHAT WILL BE USED IN THE FINAL CODE. IN THIS WAY THE RATIONALE IS SET FOR TYPES AND OBJECTS. NEXT, A LOOK AT TYPE AND OBJECT DECLARATIONS. AGAIN ACTUAL CODE RELATED TO THE EXAMPLE IS USED. CODE COMMENTS PROVIDE EXPLANATIONS OF THE ADA THUS AFTER THE COURSE IS FINISHED, THE STUDENT CAN REFER BACK TO THE COURSE NOTES WITH UNDERSTANDING. THE EXAMPLE NOW BUILDS TO ADA SUBPROGRAMS AND PARAMETERS. AT THIS POINT, THE COMPLETED CODE IS PRESENTED FOR ALL PROCEDURES AND FUNCTIONS. NEXT THESE RESOURCES ARE COLLECTED INTO AN ADA PACKAGE. ADA PROVIDES THE FACILITIES TO CREATE OUR OWN USAGE PACKAGES. THIS BUILDS AN INTUITIVE FEEL FOR THE USEFULNESS OF THE PACKAGE CONCEPT IN ADA. FINALLY, THE MAIN LOGIC PROCEDURE IS PRESENTED WHICH USES THE RESOURCES OF TWO PACKAGES. WITHIN THE MAIN PROCEDURE, A SIMPLE I/O FORMAT IS PRESENTED TO ILLUSTRATE BOTH THE ABILITY TO CREATE ONE'S OWN I/O ROUTINES, SPECIALLY TAILORED, AND TO ALSO SHOW THE USE OF THE 'GET' AND 'PUT' PROCEDURES. AS A WHOLE THE ADA EXAMPLE ILLUSTRATES A BASIC PROGRAM STRUCTURE - I.E. A MAIN DRIVER PROCEDURE USING RESOURCES FROM ONE OR MORE PACKAGES WITH THE PACKAGES IN TURN CONSISTING OF NESTED SUBPROGRAMS. AS PART OF CODING ADA, THE SYSTEM MUST BE COMPILED TO TRANSLATE THE SOURCE TO OBJECT CODE FOR EVENTUAL EXECUTION. COMPILATION AND THE PROGRAM LIBRARY ARE PRESENTED FOLLOWED BY TWO EXAMPLES OF SYSTEM CHANGE.

IT IS CRUCIAL FOR THE INSTRUCTOR TO SET UP THE PURPOSE OF THIS EXAMPLE. OTHERWISE, CONTINUAL SYNTAX QUESTIONS MAY ARISE. (THIS MAY HAPPEN ANYWAY. IF SO, GENTLY REMIND THEM OF THE PURPOSE.)

EXAMPLE 1

A SYSTEM THAT RECORDS AND TRACKS TWO-DIMENSIONAL MOVEMENT ON A RADAR SCREEN NEEDS A PROCEDURE THAT, GIVEN THE LAST POSITION RECORDED, THE CURRENT POSITION, THE TIME BETWEEN THOSE READINGS, AND A NEW TIME INTERVAL, WILL PREDICT WHERE THE NEXT POINT SHOULD OCCUR. THE PREDICTION WILL ASSUME THAT NO CHANGE IN SPEED OR DIRECTION WILL OCCUR; THE VALUE THUS OBTAINED MIGHT LATER BE COMPARED TO THE ACTUAL READING TO DETERMINE PATTERNS OF CHANGE IN EITHER FACTOR. THE TRACKING PROGRAM THUS NEEDS ACCESS TO A NEXT-POINT CALCULATION ROUTINE, WHICH SHOULD BE ASSOCIATED WITH FACILITIES TO CALCULATE THE DISTANCE BETWEEN TWO POINTS AND TO DETERMINE VELOCITY. DUE TO THE SPECIFICS OF THE SYSTEM, A VENDOR-SUPPLIED PACKAGE CONTAINING SUCH ROUTINES WOULD BE UNSUITABLE.

INSTRUCTOR NOTE

VG 732.1

2-3i

OUR EXAMPLE PROCESS

STATEMENT OF REQUIREMENTS (COMPLETED)

DECOMPOSITION OF SOLUTION

ADA IMPLEMENTATION (CODE AND COMPILATION)

CHANGES TO THE SYSTEM

INSTRUCTOR NOTES

FOR THE EXAMPLE WE ARE NOT TRYING TO SHOW THE BEST OR ONLY WAY TO APPROACH THE PROBLEM BUT RATHER TO ILLUSTRATE THE THOUGHT PROCESS INVOLVED IN ADA SYSTEMS.

WE BEGIN AT A HIGH LEVEL OF ABSTRACTION OF THE PROBLEM AND CONTINUE TO DECOMPOSE TO THE STATEMENT LEVEL.

LET US SUMMARIZE THE OBJECTS TO BE DEALT WITH AND THE OPERATIONS NEEDED TO BE PERFORMED RELATIVE TO THE OBJECTS.

A PICTURE OF A SOLUTION IS SHOWN. IT HAS BEEN DECIDED TO HAVE A MAIN PROGRAM WHICH CONTROLS THE OVERALL LOGIC FLOW OF THE SYSTEM. A SMALL PACKAGE WILL IMPLEMENT THE VECTOR CALCULATIONS. THE MAIN PROCEDURE LOGIC IS PRESENTED AS PSEUDO-CODE FOR THE MOMENT. BUT THE POSSIBLE SOLUTION MUST BE FURTHER DECOMPOSED TO MORE FULLY UNDERSTAND THE VECTOR SERVICES.

THE SAME PROCESS WOULD THEN BE DONE FOR SUCCEEDING LEVELS OF DECOMPOSITION.

DECOMPOSITION OF SOLUTION: TRACKING PROGRAM

OBJECTS

TEST POINTS

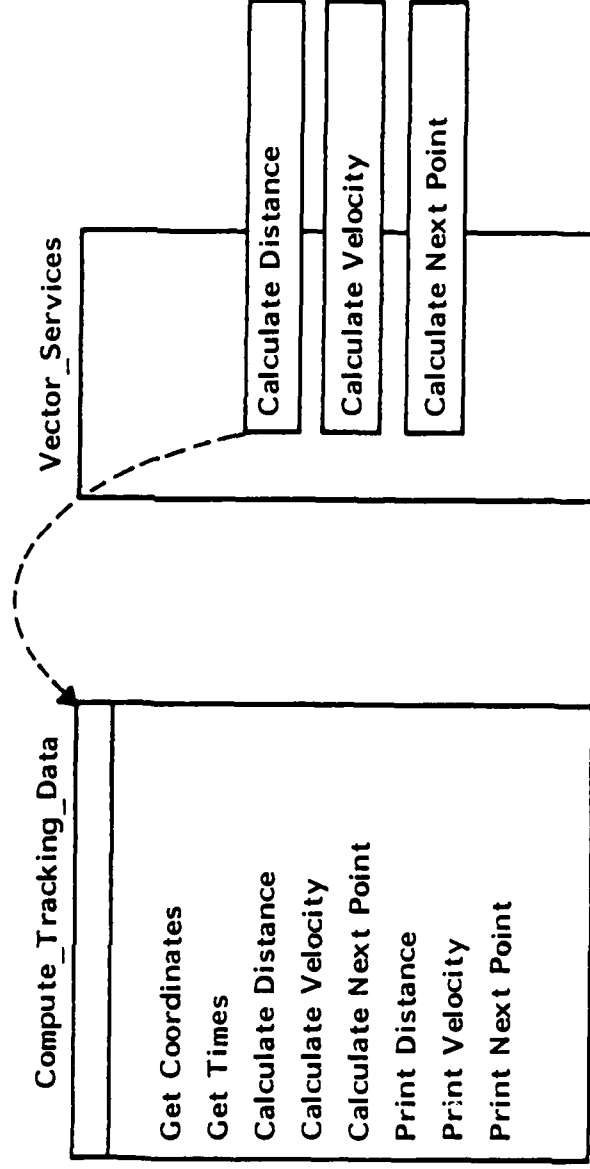
TEST TIMES

OPERATIONS

CALCULATE DISTANCE

CALCULATE VELOCITY

CALCULATE NEXT POINT



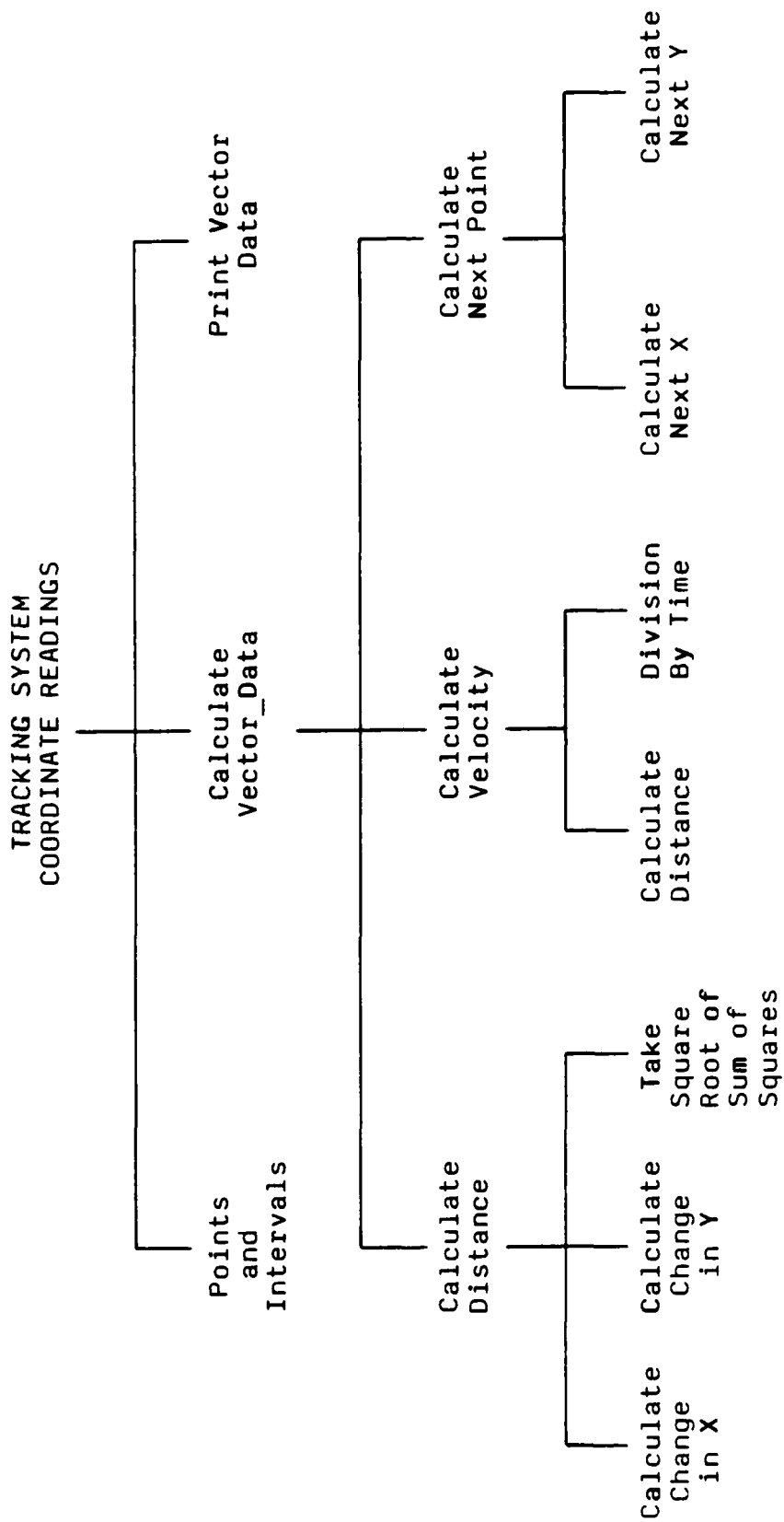
INSTRUCTOR NOTES

THE DIAGRAM SUMMARIZES THE LEVELS OF DECOMPOSITION OF THE SAMPLE DESIGN.

WE NOW TURN TO THE ACTUAL ADA CODING PHASE.

THE NAMES IN THE DIAGRAM ARE NOT THE NAMES OF THE RESULTING SUBPROGRAMS. HERE WE ARE DISCUSSING FUNCTIONS (NOT THE ADA TYPE).

DESIGN SOLUTION SUMMARY



INSTRUCTOR NOTES

THE LISTED ADA FEATURES WILL BE DISCUSSED AS PREPARATION IS MADE FOR THE CODING OF THE SOLUTION.

THESE STUDENTS WILL BE FAMILIAR WITH THE ALGORITHMS, SO DO NOT SPEND A LOT OF TIME DISCUSSING THE ALGORITHMS IN THE FOLLOWING SLIDES.

AS WE EXPRESS OUR SOLUTION FOR A TRACKING PROGRAM IN ADA, WE MUST LOOK AT:

- PACKAGES
- SUBPROGRAMS
- CONTROL STRUCTURES AND STATEMENTS
- TYPES AND DECLARATIONS

INSTRUCTOR NOTES

A LOOK AT THE RESOURCES NEEDED BY THE MAIN PROCEDURE REVEALS THAT THEY ALL PROVIDE VECTOR CALCULATION SERVICES OF SOME NATURE. THEY COULD EVEN BE USED BY SOME OTHER TRACKING SYSTEM. SO LET'S GROUP THESE RESOURCES TOGETHER IN SUCH A WAY THAT OTHER SYSTEMS CAN USE THEM. THIS IS DONE THROUGH THE ADA PROGRAM UNIT CALLED PACKAGES.

PACKAGES HAVE TWO PARTS. THE FIRST IS CALLED THE SPECIFICATION. IT TELLS WHAT KINDS OF ACTIONS OR DATA CAN BE USED.

POINT OUT THE TYPE DEFINITIONS. A DESIGN DECISION TO REPRESENT EACH POINT AS AN ARRAY WAS MADE. ARRAYS IN ADA ARE SIMILAR TO ARRAYS IN OTHER LANGUAGES; THEY WILL HAVE A SPECIFIC FORM OR TEMPLATE WHICH IS DESCRIBED IN AN ARRAY TYPE DEFINITION. THE TYPE DEFINITION PROVIDES A DESCRIPTION OF WHAT AN OBJECT OF THE TYPE WOULD LOOK LIKE-IT DOES NOT ALLOCATE ANY STORAGE.

THIS PACKAGE, CALLED `Vector_Services`, SHOWS HOW ALL OF OUR TRACKING RESOURCES CAN BE COLLECTED IN ONE LOGICAL UNIT, FOR USE BY THE MAIN PROGRAM. THE SPECIFICATION PROVIDES ALL INFORMATION NECESSARY TO USE THESE RESOURCES; WE DON'T NEED TO KNOW HOW THEY ARE IMPLEMENTED TO BE ABLE TO CODE THE MAIN PROGRAM.

PACKAGES

package Vector_Services is

TYPE DECLARATIONS

type Coordinate_Type is (X,Y);

type Point_Type is array (Coordinate_Type) of Float;

subtype Time_Type is Duration;

function Distance_Between (Last_Point, This_Point : Point_Type) return Float;

procedure Calculate_Velocity (From, To : in Point_Type;
In_Time : in Time_Type;
Velocity : out Float);

function Next_Point_After (Last_Point, This_Point : in Point_Type;
Time_Between_Last, Time_Between_Next : Time_Type)

return Point_Type;

end Vector_Services;

SPECIFICATION

INSTRUCTOR NOTES

THE ADA SYSTEM CAN NOW BE FURTHER DEVELOPED BY CODING THE MAIN LOGIC PROCEDURE. THE TRACKING RESOURCES ARE PROVIDED BY THE `Vector_Services` PACKAGE JUST SHOWN. THE 'WITH' STATEMENT MUST BE USED TO "HOOK TOGETHER" THE MAIN PROGRAM AND THE PACKAGE. THE RESOURCES FROM AN I/O PACKAGE CALLED `Text_IO` WILL ALSO BE USED.

PROCEDURE `Compute_Tracking_Data` HAS THE SAME FORMAT AS ANY OTHER PROCEDURE (EXCEPT IT HAS NO PARAMETERS). THIS SLIDE SHOWS THE DECLARATION FOR ALL DATA OBJECTS AND LOCAL ROUTINES TO BE USED IN THE STATEMENT PART. THE USE OF "IS SEPARATE" WILL BE DISCUSSED IN LATER SLIDES.

POINT OUT THE OBJECT DECLARATIONS CREATING OBJECTS OF TYPES `Point_Type` AND `Time_Type` (SHOWN ON THE PREVIOUS SLIDE) AS WELL AS OBJECTS OF THE PREDEFINED TYPE `Float`. EACH OBJECT IS GIVEN A NAME THAT REPRESENTS ITS INTENDED FUNCTION. THE TYPE TEMPLATE NAME DETERMINES NOW THE OBJECT WILL "LOOK" AND FUNCTION. (DON'T GO INTO DETAIL OR SYNTAX.)

(IF POSSIBLE, DISPLAY THIS SLIDE AND THE NEXT AT THE SAME TIME.)

MAIN PROGRAM LOGIC

DECLARATIONS:

```
with Text_IO, Vector_Services;  
use Vector_Services;  
procedure Compute_Tracking_Data is
```

OBJECT DECLARATIONS

```
Last_Point, Current_Point, Next_Point : Point_Type;  
Time_Elapsed, Time_Projected : Time_Type;  
Distance, Velocity : Float;
```

```
package Time_IO is new Text_IO.Fixed_IO (Time_Type);  
package Flt_IO is new Text_IO.Float_IO (Float);
```

```
procedure Get_Point (P : out Point_Type) is separate;  
procedure Put_Point (P : in Point_Type) is separate;
```

```
begin
```

EXECUTABLE STATEMENTS ON NEXT PAGE

```
end Compute_Tracking_Data;
```

INSTRUCTOR NOTES

THIS SLIDE SHOWS THE STATEMENT PART OF Compute_Tracking_Data.

STATEMENTS TO READ IN THE POINTS AND TIMES WITH THE SERVICES OF Text_IO; THE DESIRED INFORMATION IS CALCULATED VIA PROCEDURE AND FUNCTION CALLS; AND WE PRINT OUT RESULTS WITH THE SERVICES OF Simple_IO. NOTICE THE SUBSTITUTION OF ACTUAL PARAMETERS FOR THE FORMAL PARAMETERS OF THE SUBPROGRAM DEFINITIONS.

IF ASKED, Calculate_Velocity IS A PROCEDURE RATHER THAN A FUNCTION FOR THE PURPOSE OF COMPARING PROCEDURES AND FUNCTIONS.

MAIN PROGRAM LOGIC (CONT.)

STATEMENTS:

```
with Text_IO, Vector_Services;
use Vector_Services;
procedure Compute_Tracking_Data is

    DECLARATIONS ON PREVIOUS PAGE

begin -- Compute_Tracking_Data
    Text_IO.Put ("Enter coordinates of last position: ");
    Get_Point (Last_Point);
    Text_IO.Put ("Enter coordinates of current position: ");
    Get_Point (Current_Point);

    Text_IO.Put ("Time (in seconds) between readings : ");
    Time_IO.Get (Time_Elapsed); Text_IO.New_Line;
    Text_IO.Put ("Time (in seconds) until next reading : ");
    Time_IO.Get (Time_Projected); Text_IO.New_Line;

    Distance := Distance_Between (Last_Point, Current_Point);
    Calculate_Velocity (Last_Point, Current_Point, Time_Elapsed, Velocity);
    Next_Point := Next_Point_After (Last_Point, Current_Point,
                                    Time_Elapsed, Time_Projected);

    Text_IO.Put ("Distance between points was")
    Flt_IO.Put (Distance);
    Text_IO.Put-Line ("units.");

    Text_IO.Put ("Velocity was");
    Flt_IO.Put (Velocity);
    Text_IO.Put ("units per second.");

    Text_IO.Put ("After");
    Time_IO.Put (Time_Projected);
    Text_IO.Put ("seconds, the next point should be");
    Put_Point (Next_Point);

    end Compute_Tracking_Data;
```

INSTRUCTOR NOTES

THEN IN THE SECOND PART OF THE PROGRAM UNIT, THE BODY, IS THE ACTUAL CODE THAT PERFORMS THE RESOURCES ACTIONS. THIS SLIDE SHOWS THE FIRST TWO SUBPROGRAM BODIES -- THE OTHER TWO ARE ON THE FOLLOWING SLIDE.

NOTICE THAT PROCEDURE SQRT WAS NOT LISTED IN THE SPECIFICATION. SQRT IS A UTILITY WHICH WILL ONLY BE USED BY THE ALGORITHM Distance_Between. BY PLACING IT IN THE PACKAGE BODY, WE ENSURE THAT NO UNAUTHORIZED TAMPERING OF THE DATA CAN BE DONE.

BRIEFLY DISCUSS THE INDICATED CONTROL STRUCTURES, POINTING OUT RESERVED WORDS (UNDERLINING MAY BE HELPFUL). DO NOT GET BOGGED DOWN IN SYNTAX; FOCUS ON GENERAL STRUCTURE AND FUNCTION. POINT OUT NESTED CONTROL STRUCTURE, WITH INDENTATION SHOWING LOGICAL NESTING.

THE while LOOP IS AN ITERATIVE CONTROL STRUCTURE, ALLOWING REPETITION OF SOME SEQUENCE OF ACTION WHILE SOME CONDITION IS PRESENT. THE OTHER ITERATIVE CONTROL STRUCTURE IS THE for LOOP (NOT SHOWN), WHICH ALLOWS REPETITION FOR A SPECIFIED NUMBER OF TIMES.

THE VECTOR PACKAGE BODY

```

package body Vector_Services is

    function Sqrt (X : Float) return Float is
        Epsilon : constant := 0.000001;
        Root : Float := 1.0;
    begin
        if X = 0.0 then
            return 0.0;
        else
            Root := (X/Root + Root) / 2.0;
            while abs (X/Root**2 - 1.0) >= Epsilon
            loop
                Root := (X/Root + Root) / 2.0;
            end loop;
            return Root;
        end if;
    end Sqrt;

    function Distance_Between (Last_Point, This_Point : Point_Type) return Float is
        Dx, Dy : Float;
    begin
        Dx := abs (This_Point(X) - Last_Point(X));
        Dy := abs (This_Point(Y) - Last_Point(Y));
        return ( Sqrt( Dx**2 + Dy**2) );
    end Distance_Between;

```

-- LOCAL DECLARATIONS

IF/THEN/ELSE CONTROL STRUCTURE

LOOP CONTROL STRUCTURE

INSTRUCTOR NOTES

DATA DECLARATIONS, CONTROL STRUCTURES, AND ASSIGNMENT STATEMENTS ARE BUILT INTO FUNCTIONAL (EXECUTABLE) STRUCTURES KNOWN AS SUBPROGRAMS IN ADA. SUBPROGRAMS HAVE 2 FORMS - PROCEDURES AND FUNCTIONS, SIMILAR TO OTHER LANGUAGES.

INDICATE PROCEDURE AND FUNCTION TEMPLATE STRUCTURE BY UNDERLINING RESERVED WORDS.

A PROCEDURE BEGINS EXECUTION THROUGH A PROCEDURE CALL (SHOWN IN MAIN PROCEDURE BODY), WHICH IS A STATEMENT. A FUNCTION CALL IS AN EXPRESSION (RETURNS A VALUE); THUS EVERY FUNCTION MUST SPECIFY A RETURN TYPE AND MUST EXPLICITLY RETURN A VALUE VIA A RETURN STATEMENT.

POINT OUT THE PARAMETER LISTS AND MODE INDICATIONS. A PARAMETER OF MODE IN IS PASSED TO THE SUBPROGRAM BUT CANNOT BE MODIFIED IN IT; AN OUT PARAMETER IS ONE THAT RETURNS A VALUE ASSIGNED TO IT IN THE SUBPROGRAM. A THIRD MODE, IN OUT, INDICATES A PARAMETER THAT IS PASSED IN, MODIFIED, AND PASSED OUT AGAIN. A FUNCTION PARAMETER MAY BE OF MODE IN ONLY.

PACKAGE BODY (CONT.)

```

procedure Calculate_Velocity (From, To : in Point_Type); -- FORMAL DEFINITION OF
    In Time : in Time_Type; -- Calcualte Velocity with
    Velocity : out Float) is -- FORMAL (DUMMY) PARAMETER LIST
begin
    Velocity := Distance_Between(From, To)/Float(In_Time);
end Calculate_Velocity;

function Next_Point_After (Last_Point, This_Point : in Point_Type;
    Time_Between_Last, Time_Between_Next : Time_Type)
    return Point_Type is
    --_VALUE RETURNED IS OF Point_Type
    Next_Point : Point_Type;
begin
    if Time_Between_Last = 0 then
        return This_Point;
    else
        Next_Point(X) := Last_Point(X) + Float(Time_Between_Next/Time_Between_Last)
            * abs (This_Point(X) - Last_Point(X));
        Next_Point(Y) := Last_Point(Y) + Float(Time_Between_Next/Time_Between_Last)
            * abs (This_Point(Y) - Last_Point(Y));
        return Next_Point;
    end if;
end Next_Point_After;

end Vector_Services;

```

INSTRUCTOR NOTES

CODING OF THE ADA SYSTEM IS COMPLETED. NEXT THE TOPIC OF COMPILATION IN ADA IS DISCUSSED.

EXPECT SOME QUESTION ABOUT CURRENT IMPLEMENTATIONS OF ADA, SUCH AS SPEED OF EXECUTION, SPEED OF COMPILATION, ETC. TELL THE STUDENTS (IF YOU DO NOT KNOW THESE FIGURES) THAT ADA COMPILERS ARE TOO NEW TO ADEQUATELY ANSWER THESE QUESTIONS.

BREAK HERE FOR 15 MINUTES.

WE NOW NEED TO COMPILE OUR ADA SYSTEM

VG 732.1

2-12

INSTRUCTOR NOTES

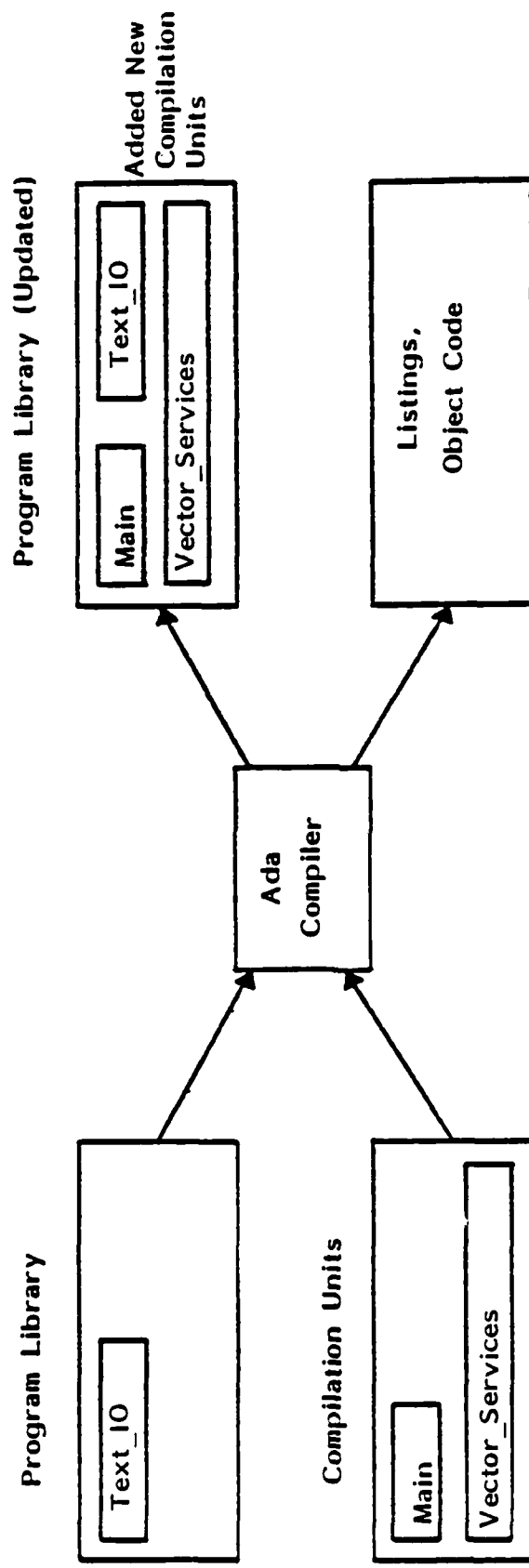
COMPILATION UNITS ARE PARTS OF ADA CODE THAT THE LANGUAGE SAYS CAN BE SUBMITTED BY THEMSELVES TO AN ADA COMPILER.

COMPILATION CONSISTS OF SUBMITTING OUR COMPILATION UNITS PLUS THE PROGRAM LIBRARY WHICH IS A FILE THAT WILL CONTAIN CERTAIN INFORMATION ABOUT A UNIT THAT SUBSEQUENT COMPILER SUBMISSION WILL NEED. ONCE COMPILED, THE SUBMITTED COMPILATION UNITS ARE ADDED TO THE PROGRAM LIBRARY.

MAIN = Compute_Tracking_Data ON FOLLOWING CHARTS
(DUE TO SPACE LIMITATIONS).

COMPILATION OF OUR TRACKING SYSTEM

- SUBMIT ALL PROGRAM PARTS AT ONE TIME:



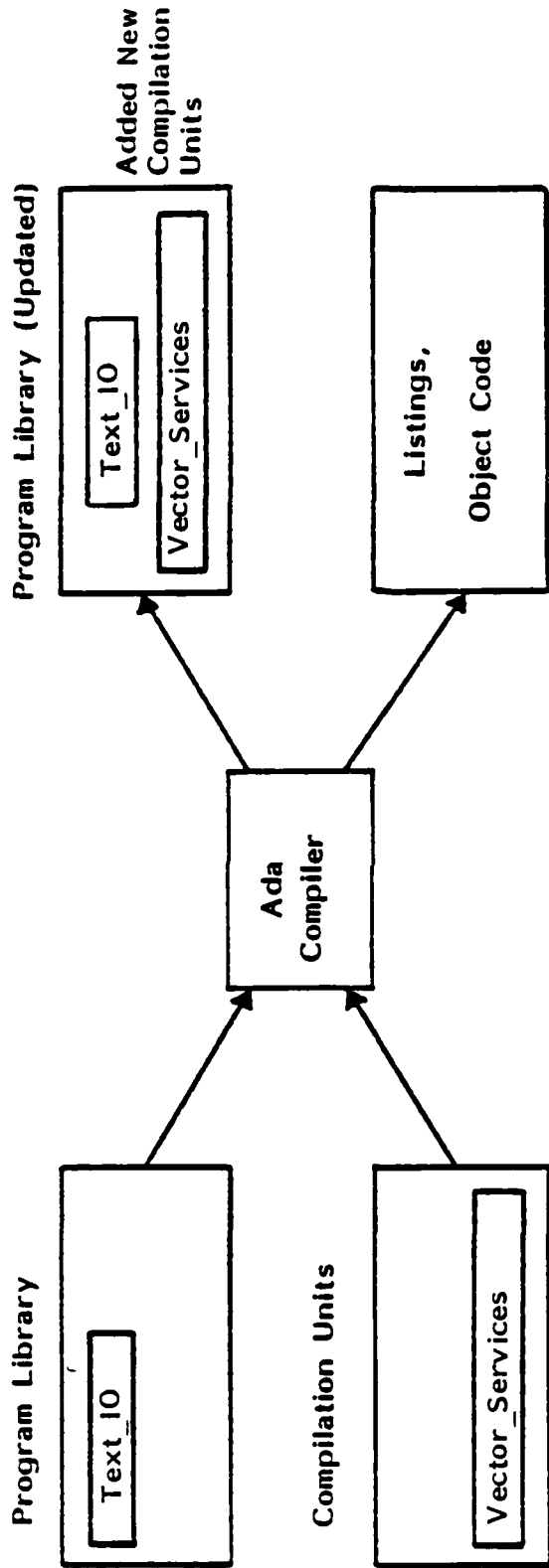
INSTRUCTOR NOTES

INSTEAD OF SUBMITTING ALL OUR PROGRAM PARTS AT ONE TIME, WE COULD SUBMIT THEM SEPARATELY. LET'S SAY PROGRAMMER 1 CODED OUR `Vector_Services` PACKAGE. INSTEAD OF WAITING FOR PROGRAMMER 2, WHO WILL HAVE HER CODE COMPLETED LATER, WE CAN COMPILE THE `Vector_Services` PACKAGE. THE COMPILER WILL ADD THE NECESSARY INFORMATION ABOUT THE PACKAGE TO THE PROGRAM LIBRARY.

ALTERNATE COMPILATION OF OUR HOBBIT SYSTEM

- SUBMIT PROGRAM PARTS (COMPILATION UNITS) SEPARATELY:

RUN 1



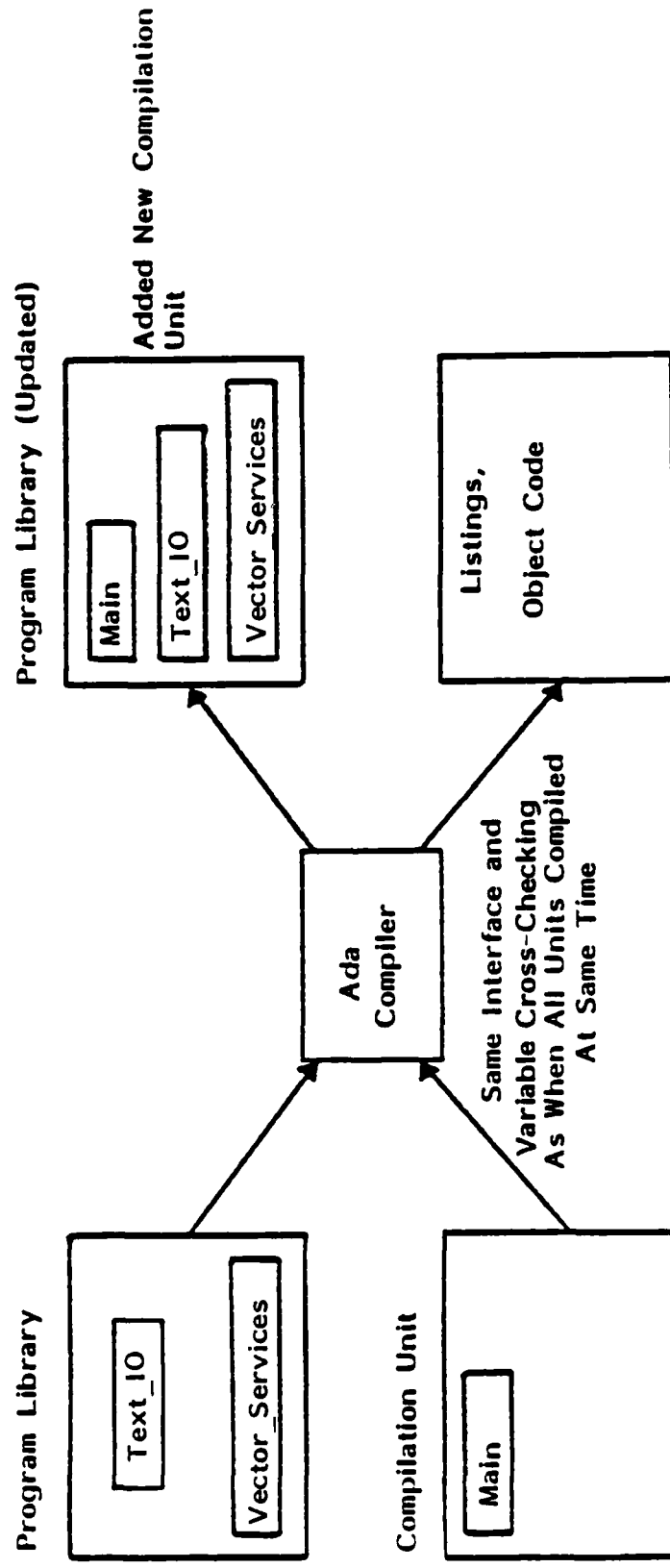
INSTRUCTOR NOTES

WHEN PROGRAMMER 2 IS FINISHED, WE THEN SUBMIT OUR PROCEDURE MAIN PLUS THE PROGRAM LIBRARY TO THE ADA COMPILER. WITH THE INFORMATION CONTAINED IN THE PROGRAM LIBRARY, THE COMPILER CAN DO THE SAME INTERFACE AND VARIABLE CROSS-CHECKING BETWEEN MAIN AND THE PACKAGE - JUST AS IF THEY HAD BEEN COMPILED AT THE SAME TIME. THIS THEN IS AN EXAMPLE OF SEPARATE COMPILATION.

1. 2. 3. 4. 5. 6. 7. 8. 9. 10. 11. 12. 13. 14. 15. 16. 17. 18. 19. 20. 21. 22. 23. 24. 25. 26. 27. 28. 29. 30. 31. 32. 33. 34. 35. 36. 37. 38. 39. 40. 41. 42. 43. 44. 45. 46. 47. 48. 49. 50. 51. 52. 53. 54. 55. 56. 57. 58. 59. 60. 61. 62. 63. 64. 65. 66. 67. 68. 69. 70. 71. 72. 73. 74. 75. 76. 77. 78. 79. 80. 81. 82. 83. 84. 85. 86. 87. 88. 89. 90. 91. 92. 93. 94. 95. 96. 97. 98. 99. 100.

ALTERNATE COMPILATION (Continued)

RUN 2



THIS WAY IS CALLED SEPARATE COMPILATION.

INSTRUCTOR NOTES

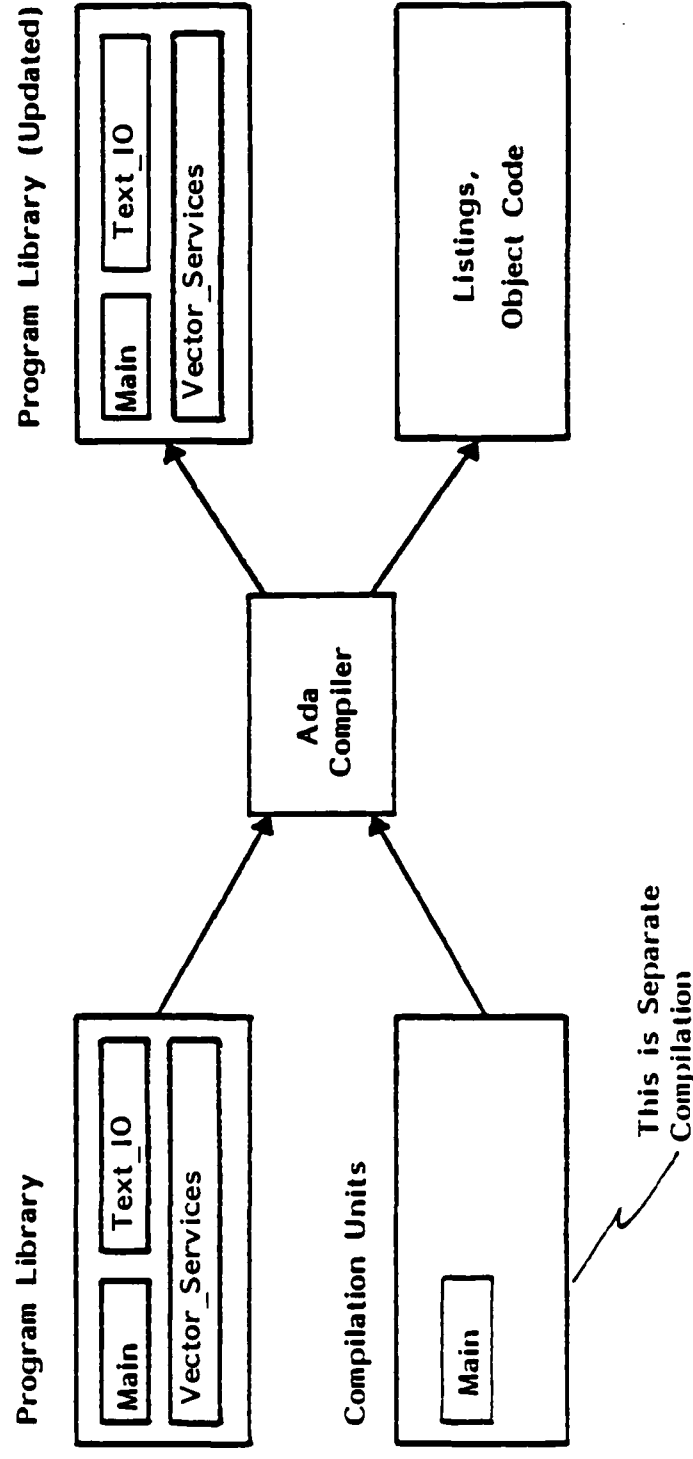
THE NATURE OF LARGE SYSTEMS IS CONTINUAL CHANGE. WE NEXT LOOK AT HOW THAT CAN AFFECT OUR SOLUTION.

THE GOAL OF THIS SLIDE IS TO ILLUSTRATE ONE OF THE GREAT ADVANTAGES OF ADA - THE PACKAGE - FOR LOCALIZATION OF EFFECT OF CHANGES.

ASK THE CLASS: IF WE WANT TO CHANGE THE OUTPUT FORMATS, WHAT DO WE NEED TO CHANGE?

CHANGES TO THE SYSTEM: MAIN PROCEDURE

WE NEED TO CHANGE ONE OF THE PRINTOUT FORMATS. SINCE THE PACKAGE WORRIES ABOUT ALL AND ONLY THE VECTOR CALCULATIONS, THE PACKAGE NEED NOT BE CHANGED OR RECOMPILED.



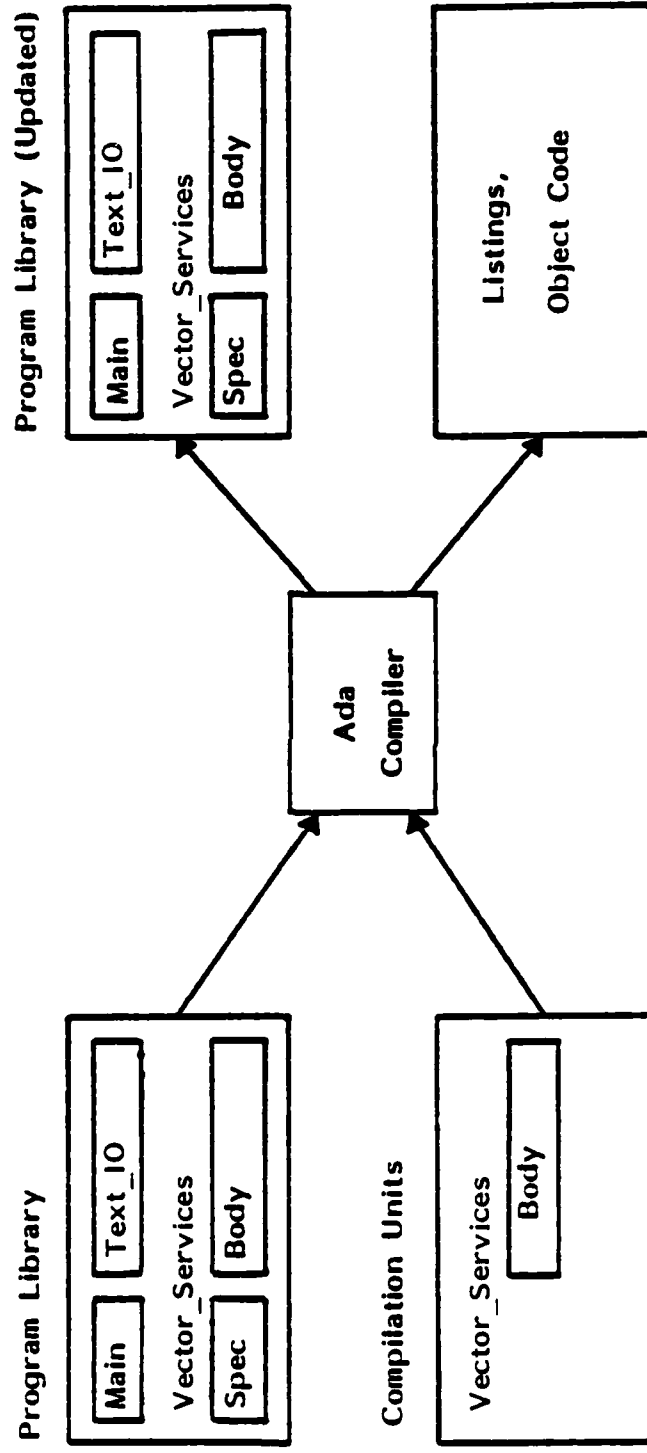
INSTRUCTOR NOTES

ANOTHER EXAMPLE OF EASE OF CHANGE.

POINT OUT THAT NEITHER PROCEDURE MAIN NOR THE PACKAGE SPECIFICATION FOR
Vector_Services NEED TO BE RECOMPILED.

CHANGES TO THE SYSTEM: PACKAGE BODY

WE FIND A BETTER ALGORITHM FOR ONE OF OUR Vector ROUTINES. SINCE WE COLLECTED OUR ROUTINES IN A PACKAGE, WE CAN MAKE THE CHANGE TO THE PACKAGE BODY Vector_Service WITHOUT REQUIRING ANY CHANGES TO THE MAIN PROCEDURE OR THE PACKAGE SPECIFICATION FOR Vector_Services.



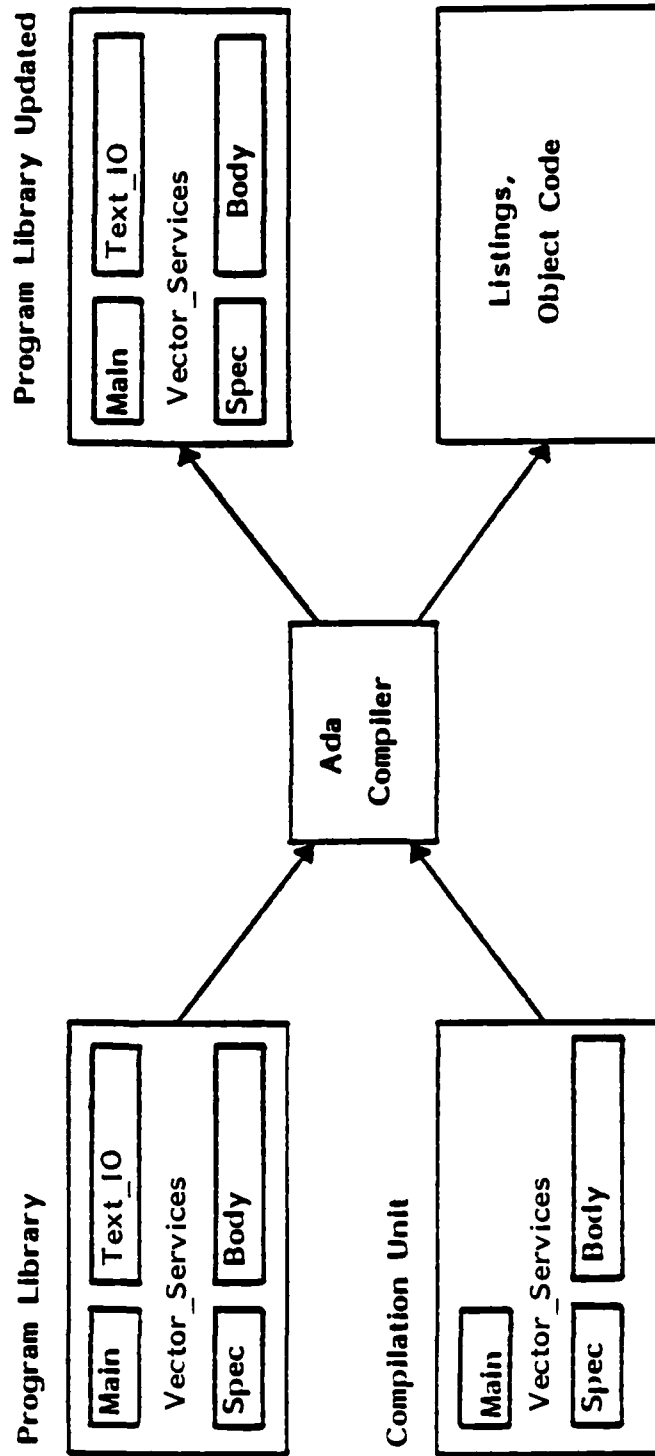
Separate
Compilation

INSTRUCTOR NOTES

CHANGES TO THE SYSTEM: ADDING A ROUTINE

WE WANT TO ADD A ROUTINE TO COMPUTE THE ANGLE OF THE Vector SINCE WE COLLECTED OUR Vector ROUTINES IN A PACKAGE, WE WANT TO ADD THIS ROUTINE TO THE PACKAGES SPECIFICATION AND BODY OF Vector_Services. WE MODIFIED Vector_Services AND OUR MAIN PROCEDURE DEPENDS ON THOSE RESOURCES.

AS A RESULT WE MUST ALSO RECOMPILE THE MAIN PROCEDURE.



WE'D LIKE TO KEEP OUR SYSTEM UNDERSTANDABLE AND READABLE. IN PURSUING THIS GOAL, WE
RETURN TO THE PACKAGE BODY OF Vector_Services.

INSTRUCTOR NOTES

IN REALITY THE '...' IS A NUMBER OF LINES OF CODE. AS SHOWN ON TWO PREVIOUS SLIDES IN THIS FORM, WE REALLY CAN'T SEE THE STRUCTURE OF THE BODY OR EASILY FIND A SECTION OF CODE WE MAY BE INTERESTED IN.

PACKAGE BODY STRUCTURE

- THIS FORMAT CAN BE CONFUSING IN REALITY

```
package body Vector_Services is
function Sqrt (X: Float) return Float is
    Epsilon: constant := 0.000001;
    Root: Float := 1.0;
begin
    if X = 0.0 then
        return 0.0;
    else
        Root := (X/Root + Root)/2.0;
        while abs (X/Root **2 - 1.0) >= Epsilon
            loop
                Root := (X/Root + Root)/2.0;
            end loop;
        end loop;
        return Root;
    end if;
end Sqrt;
function Distance_Between (Last_Point, This_Point : Point_Type)
    return Float is ... begin ... end;
procedure Calculate_Velocity (From, To : in Point_Type;
    In_Time : in Time_Type;
    Velocity : out Float) is ... begin ... end;
function Next_Point_After (Last_Point, This_Point : in Point_Type;
    Time_Between_Last, Time_Between_Next : Time_Type)
    return Point_Type is ... begin ... end;
end Vector_Services;
```

INSTRUCTOR NOTES

ADA ALLOWS US TO CAPTURE THE INITIAL STRUCTURE AND COMPOSITION OF THE PACKAGE BODY THROUGH STUBBING.

'IS SEPARATE' JUST SAYS TO THE COMPILER, "YOU WILL FIND THE ACTUAL CODE FOR THIS SUBPROGRAM IN A SEPARATE PLACE FROM THE PARENT (OR CONTAINING) ADA UNIT".

IN CONCEPT, STUBBING IS SIMILAR TO SUBROUTINES IN FORTRAN, ASSEMBLY LANGUAGE, JOVIAL.

ALTERNATIVE PACKAGE BODY STRUCTURE

```
package body Vector_Services is

  function Sqrt (X : Float) return Float is separate; -- A STUB

  function Distance_Between (Last_Point, This_Point : Point_Type)
    return Float is separate;

  procedure Calculate_Velocity (From, To : in Point_Type;
    In_Time : in Time_Type;
    Velocity : out Float) is separate;

  function Next_Point_After (Last_Point, This_Point : in Point_Type;
    Time_Between_Last, Time_Between_Next : Time_Type)
    return Point_Type is separate;

end Vector_Services;
```


- IN ADDITION, FOR EACH 'SEPARATE' SUBPROGRAM (SUBUNIT) WE INDICATE THE PARENT UNIT

```

separate (Vector_Services)      -- We Add This Line
function Next_Point_After (Last_Point, This_Point : in Point_Type;
    Time_Between_Last, Time_Between_Next : Time_Type)
    return Point_Type is

    Next_Point : Point_Type;

begin

    if Time_Between_Last = 0 then
        return This_Point;
    else
        Next_Point(X) := Last_Point(X) + Float(Time_Between_Next/Time_Between_Last)
            * abs (This_Point(X) - Last_Point(X));
        Next_Point(Y) := Last_Point(Y) + Float(Time_Between_Next/Time_Between_Last)
            * abs (This_Point(Y) - Last_Point(Y));
        return Next_Point;
    end if;
end Next_Point_After;

```

INSTRUCTOR NOTES

THESE ARE THE SUBUNITS STUBBED OUT OF THE MAIN PROCEDURE. NOTE THAT THIS CODE WOULD ADD CONSIDERABLE BULK TO THE MAIN PROCEDURE BODY IF USED INLINE, WHILE CONTRIBUTING LITTLE TO THE LOGICAL STRUCTURE. STUBBING OUT THESE ROUTINES ALLOWS EASY MODIFICATION OF I/O FORMAT.

MORE SUBUNITS

```
separate (Compute_Tracking_Data)
procedure Get_Point (P : out Point_Type) is
begin
    Text_IO.Put (" X = ");
    Flt_IO.Get (P(X));
    Text_IO.Put (" Y = ");
    Flt_IO.Get (P(Y));
    Text_IO.New_Line;
end;
```

```
separate (Compute_Tracking_Data)
procedure Put_Point (P : in Point_Type) is
begin
    Text_IO.Put ("(");
    Flt_IO.Put (P(X));
    Text_IO.Put (",");
    Flt_IO.Put (P(Y));
    Text_IO.Put (")");
end;
```

INSTRUCTOR NOTES

THE FOLLOWING EXAMPLE SPANS 3 SLIDES AND STEPS THROUGH ONE POSSIBLE WAY TO SEPARATELY
COMPILE THE SYSTEM WE'VE JUST SEGMENTED.

SPECS MUST BE COMPILED BEFORE BODIES.

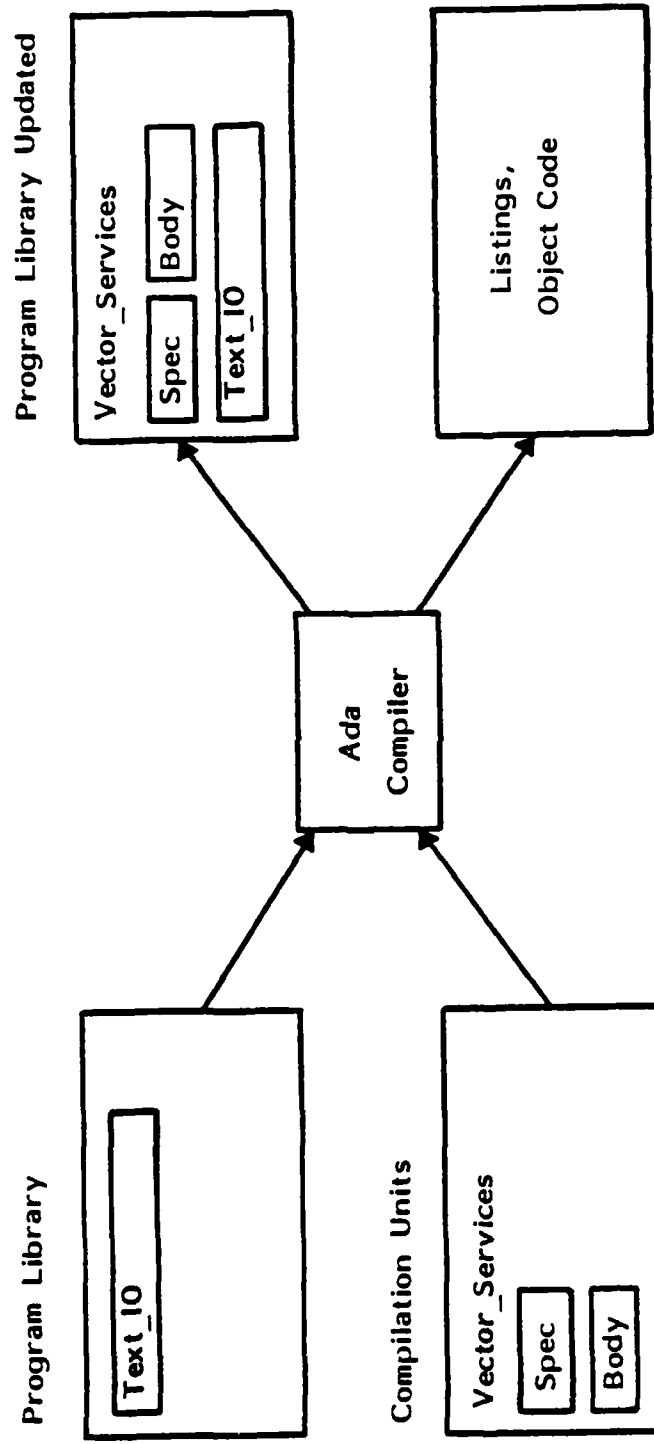
VG 732.1

2-241

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

TO SEPARATELY COMPILE OUR SYSTEM WITH SUBUNITS, AN EXAMPLE:

RUN 1

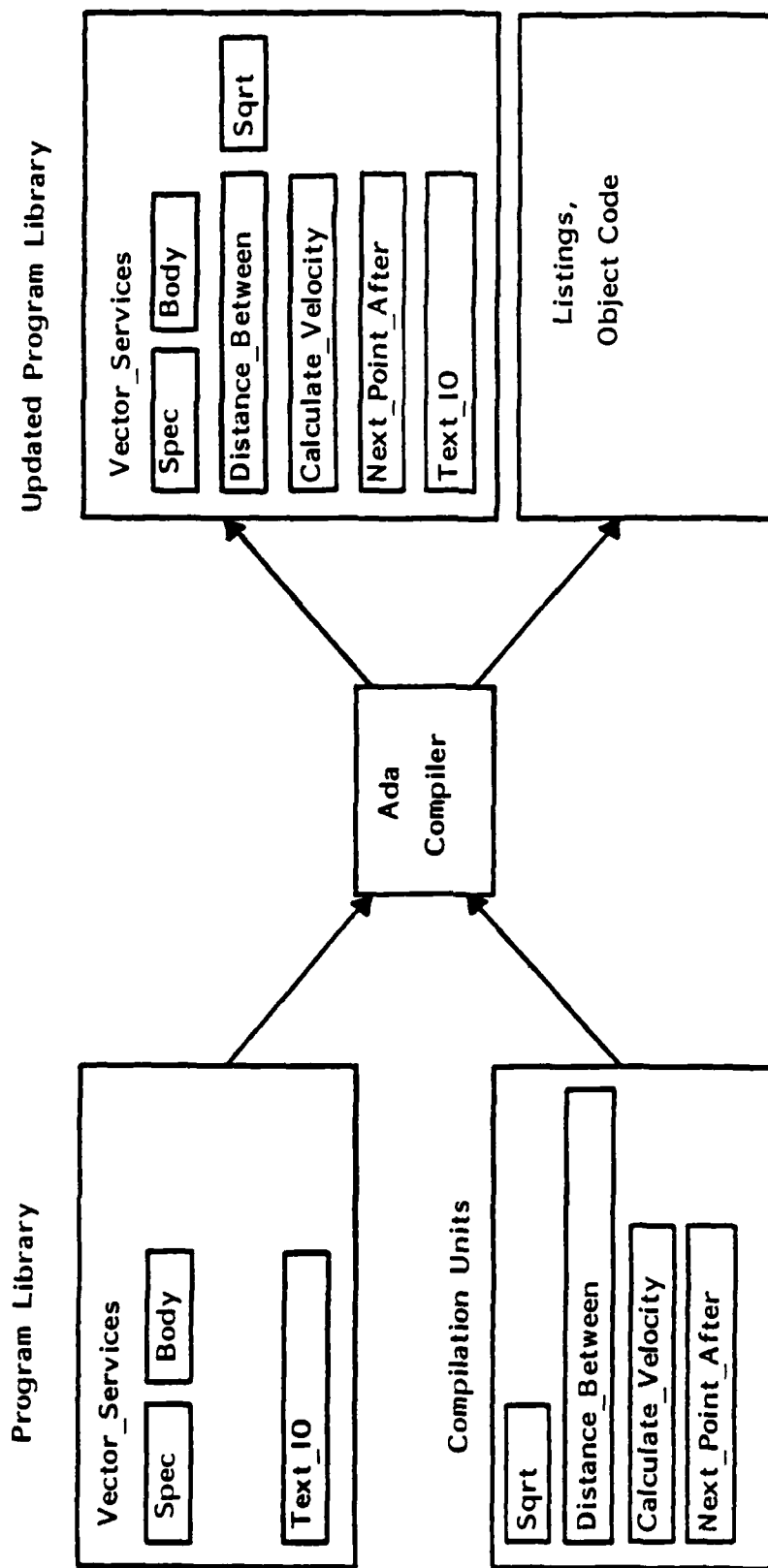


INSTRUCTOR NOTES

FOR OUR EXAMPLE, WE WILL COMPILE THE PACKAGE SUBUNITS AND ADD THEM TO THE PROGRAM LIBRARY.

ALL FOUR SUBUNITS NEED NOT BE COMPILED AT THE SAME TIME. HOWEVER, ANY SUBUNIT THAT DEPENDS ON ANOTHER MUST BE COMPILED AFTER THE ONE UPON WHICH IT DEPENDS. FOR EXAMPLE, Distance_Between MUST BE COMPILED BEFORE Calculate_Velocity.

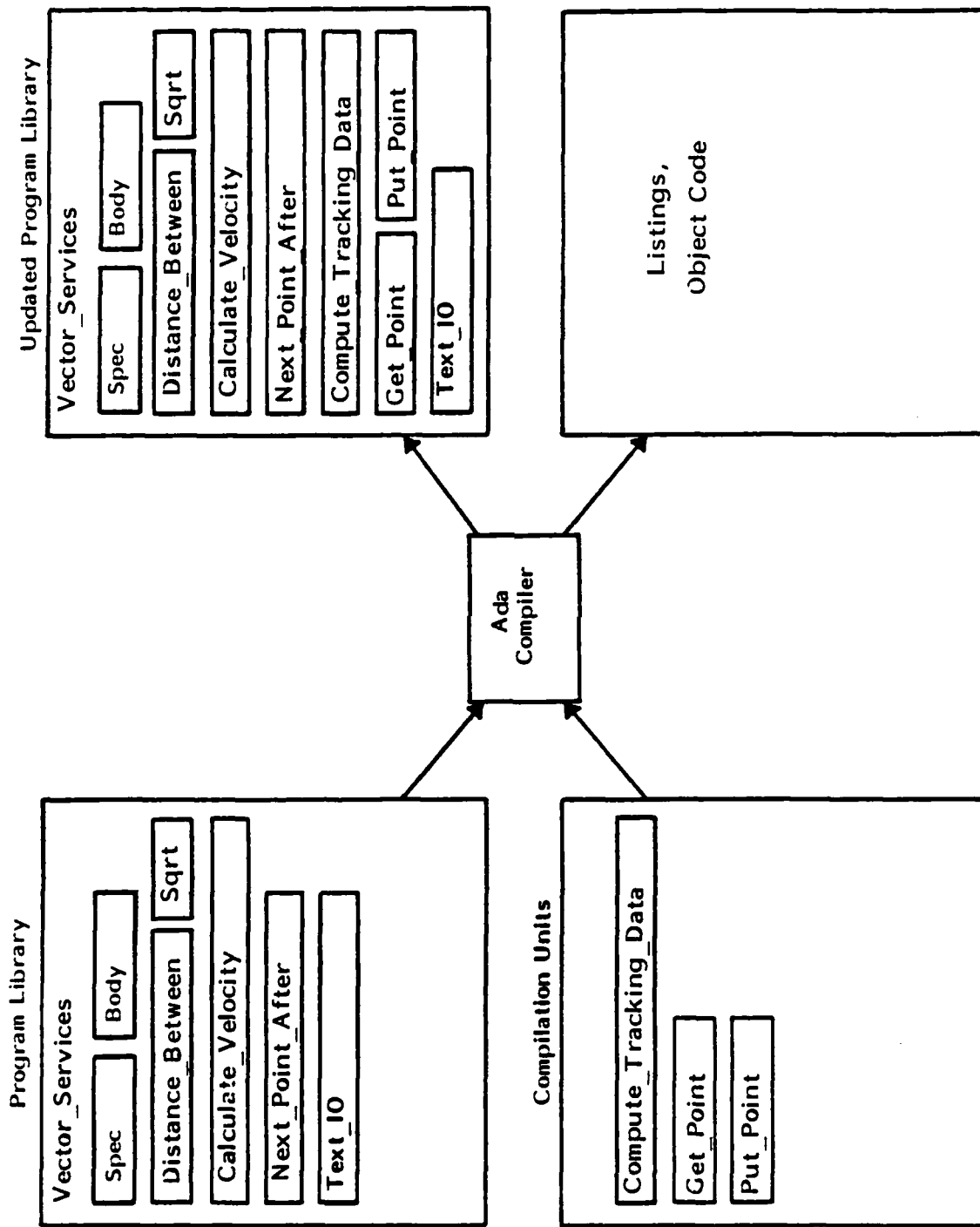
RUN 2



INSTRUCTOR NOTES

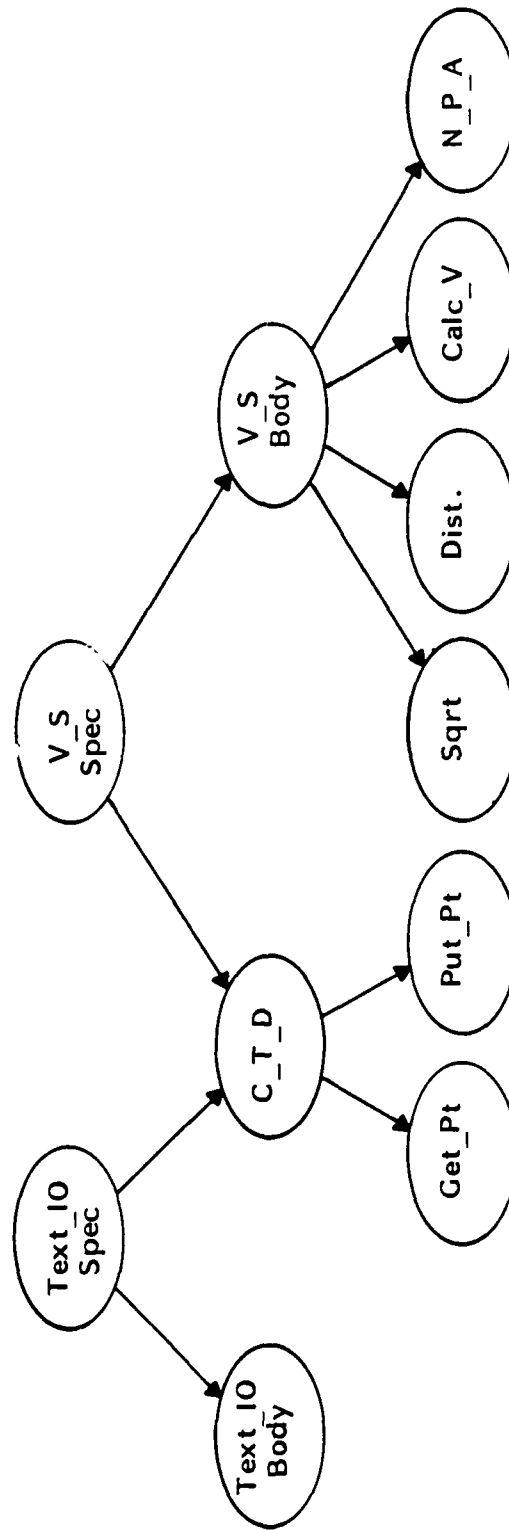
ONCE ALL THE RESOURCE PIECES NEEDED BY THE MAIN PROCEDURE ARE IN PROGRAM LIBRARY, WE CAN
COMPILE Compute_Tracking_Data AND ITS SUBUNITS.

RUN 3



INSTRUCTOR NOTES

HERE IS THE DEPENDENCY DIAGRAM



ALL POSSIBLE ORDERINGS CAN BE DERIVED FROM THE ABOVE DIAGRAM.

AD-A165 352

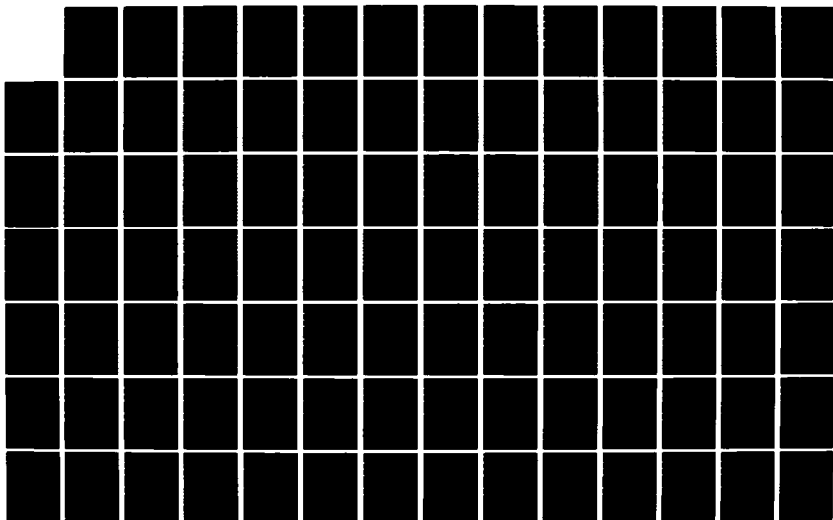
ADA (TRADEMARK) TRAINING CURRICULUM: ADA TECHNICAL
OVERVIEW L102 TEACHER'S GUIDE(U) SOFTECH INC WALTHAM MA
1986 DAA877-83-C-K506

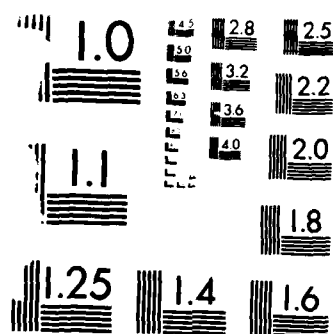
2/3

UNCLASSIFIED

F/G 5/9

NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

IN-CLASS EXERCISE

SUGGEST OTHER COMPILATION ORDER POSSIBILITIES.

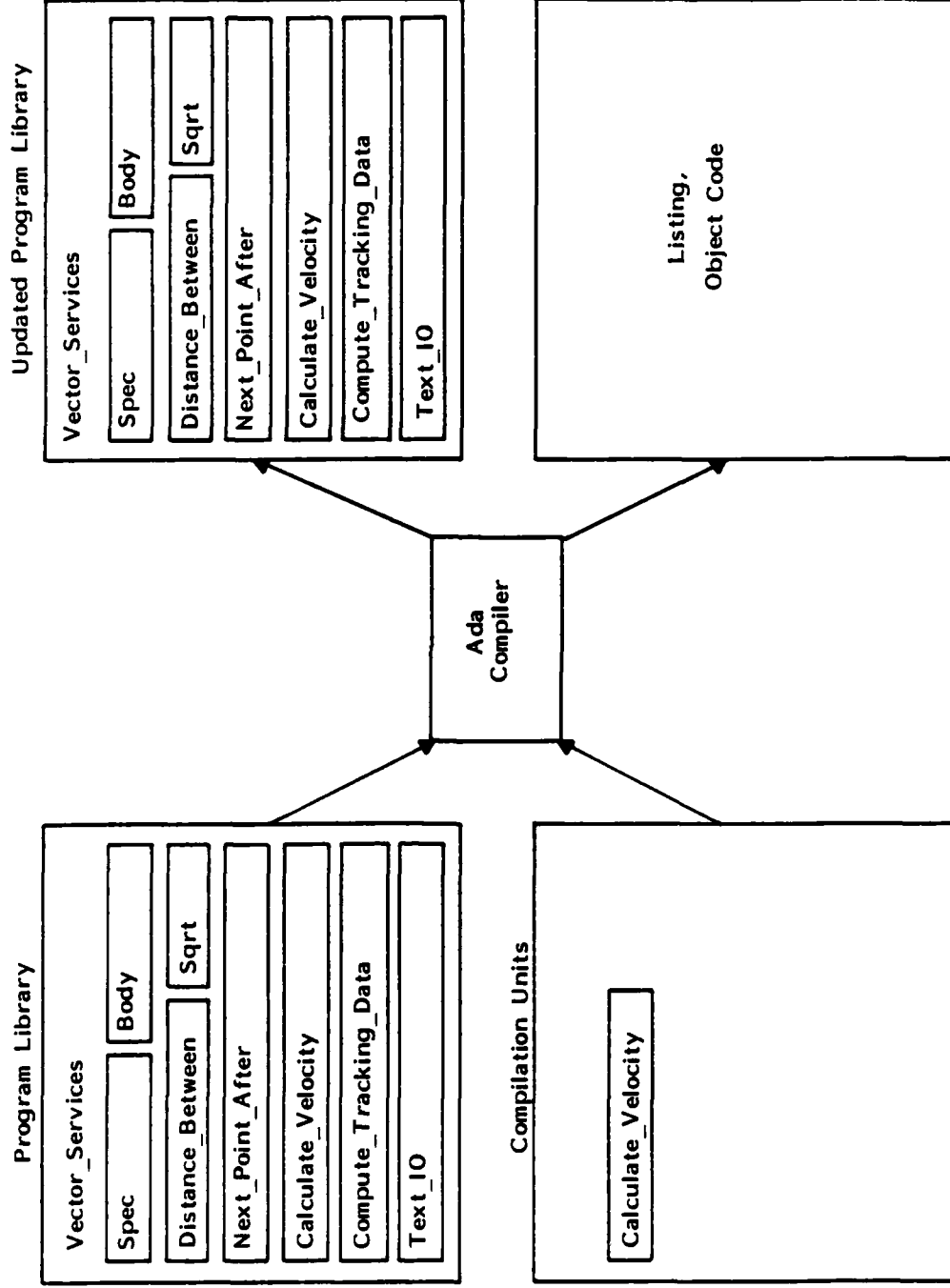
INSTRUCTOR NOTES

SINCE THE PACKAGE SPECIFICATION IS NOT CHANGED, WE DO NOT NEED TO RECOMPILE THE MAIN PROCEDURE.

NOTE, HOW WE REDUCE THE AMOUNT OF MODIFICATION AND RECOMPILING OF THE SYSTEM. ALSO SEVERAL PROGRAMMERS COULD BE WORKING SIMULTANEOUSLY.

CHANGES TO THE SYSTEM: A SUBUNIT

WE MODIFY ONE FUNCTION IN THE PACKAGE BODY.



INSTRUCTOR NOTES

THIS SECTION FORMALIZES ADA PROGRAM STRUCTURE FROM OUR PREVIOUS EXAMPLE.

ALLOW 15 MINUTES FOR THIS SECTION.

Section 3

Summary of Ada Program Structure

INSTRUCTOR NOTES

VG 732.1

3-11

TOPIC OUTLINE

BACKGROUND AND RATIONALE FOR ADA

WRITING AN ADA PROGRAM FROM BEGIN TO END

SUMMARY OF ADA PROGRAM STRUCTURE

ADA THROUGH EXAMPLE

LARGE SYSTEM DEVELOPMENT

SUMMARY OF ADA FEATURES

FOR MORE INFORMATION

INSTRUCTOR NOTES

A LIST OF THE THREE STRUCTURAL BUILDING BLOCKS OF ANY ADA SYSTEM.

BRIEFLY SAY WHAT EACH DOES IN ADA, E.G. PACKAGES PROVIDE A MEANS TO COLLECT RELATED DATA AND ALGORITHMS, SUBPROGRAMS ARE SIMILAR TO OTHER LANGUAGES - THEY PROVIDE OUR ALGORITHMS, AND TASKS PROVIDE MECHANISMS FOR REAL TIME PROCESSING.

PROGRAM UNITS

ADA SYSTEMS CAN CONSIST OF COMBINATIONS OF:

- PACKAGES
- SUBPROGRAMS
 - PROCEDURES
 - FUNCTIONS
- TASKS
- GENERICS

INSTRUCTOR NOTES

THE SEPARATION OF THE SPECIFICATION FROM THE BODY (THE WHAT FROM THE HOW) IS WHAT GIVES US THE RELIABILITY AND MAINTAINABILITY POINTS OF THE SLIDE. REALLY EXPLAIN THE SPECIFICATION AND BODY AND WHY IT'S IMPORTANT.

INTERFACE ERRORS ARE ONE OF THE MAJOR PROBLEMS IN INTEGRATING MODULES IN LARGE SYSTEMS. WITH THE SPECIFICATION INFORMATION, THE COMPILER CAN PERFORM VALIDITY CHECKS AT COMPILE-TIME RATHER THAN INTEGRATION TIME. IN OTHER WORDS, YOU CAN TEST THE INTERFACES OF THE DESIGN AS A WHOLE BEFORE CODING ANY OF THE ALGORITHMS. IT IS MORE COST EFFECTIVE TO CORRECT ERRORS AT THIS POINT THAN AT INTEGRATION AND TESTING.

SPECIFICATIONS CAN BE VIEWED AS LOGICAL INTERFACES.

PROGRAM UNIT STRUCTURE

ALL PROGRAM UNITS HAVE A SIMILAR FORM

- SPECIFICATION

DESCRIBES WHAT THE PROGRAM UNIT DOES

THIS INFORMATION IS 'VISIBLE' TO (CAN BE REFERENCED BY) THIS AND
OTHER PROGRAM UNITS

- BODY

DETAILS HOW THE PROGRAM UNIT IMPLEMENTS AN ALGORITHM OR STRUCTURE

THIS INFORMATION IS 'HIDDEN' FROM (CANNOT BE DIRECTLY REFERENCED BY)
OTHER PROGRAM UNITS

RELIABILITY INCREASED BECAUSE INTERFACE (SPECIFICATION) ERRORS CAN BE EASILY DETECTED

MAINTAINABILITY INCREASED BECAUSE CHANGES TO THE IMPLEMENTATION (BODY) CAN BE DONE
WITHOUT AFFECTING USER PROGRAM UNITS

INSTRUCTOR NOTES

WE SAW EXAMPLES OF SEPARATE COMPILATION IN THE TRACKING SYSTEM OF SECTION 2.

VG 732.1

3-4i

SEPARATE COMPILATION

BECAUSE OF THE SPECIFICATION/BODY DISTINCTION IN PROGRAM UNITS, LARGE ADA PROGRAMS MAY BE BROKEN INTO PIECES WHICH ARE COMPILED SEPARATELY.

- A COMPILATION CONSISTS OF ONE OR MORE COMPILATION UNITS WHICH ARE SUBMITTED TOGETHER TO THE ADA COMPILER.
- COMPILATION UNITS MAY BE:
 - package specification
 - subprogram specification
 - package body
 - subprogram body
 - subunit

INSTRUCTOR NOTES

EMPHASIZE THAT STUBBING AND SUBUNITS ARE INDIVISABLE.

POINT OUT THAT IT REPRESENTS A MECHANISM FOR TOP DOWN DEVELOPMENT OF LARGE SYSTEMS USING TEAMS OF PROGRAMMERS.

- ALLOWS PROJECTS TO BE SPLIT AMONG SEVERAL PROGRAMMERS, EACH COMPILING THEIR OWN CODE.
- INCREASES READABILITY BY ONLY INCLUDING SPECS OF NESTED SUBPROGRAMS.

SUBUNITS

- THE "TOP DOWN" APPROACH TO SEPARATE COMPILE INVOLVES USING BODY STUBS AND SUBUNITS.
- AT THE POINT WHERE A SUBPROGRAM BODY OR PACKAGE BODY WOULD NORMALLY APPEAR IN A COMPILE, A BODY STUB MAY BE USED INSTEAD:

```
procedure Subprogram_Name is separate;
```

THIS IMPLIES THAT THE ACTUAL BODY WILL BE SUPPLIED IN A SEPARATE SUBUNIT.

- THE BODY IS SUPPLIED WITH A PREFIX INDICATING OR NAMING THE COMPILE UNIT WHERE THE CORRESPONDING BODY STUB APPEARED

```
separate (Parent_Unit)      -- note no semicolon  
procedure Subprogram_Name is -- body
```

- ALTHOUGH THE SUBUNIT IS SEPARATELY COMPILED, THE EFFECT IS EXACTLY AS IF THE ACTUAL BODY WERE GIVEN AT THE POINT OF THE BODY STUB

INSTRUCTOR NOTES

CAUTION: IT IS VERY IMPORTANT THAT THE INSTRUCTOR BE QUITE FAMILIAR WITH THE FOLLOWING SAMPLE BEFORE PRESENTING THE MATERIAL. ALSO, DO NOT ALLOW THE STUDENTS TO DWELL ON SYNTAX. AT THIS OVERVIEW LEVEL, WE WANT TO CONCENTRATE ON THE CONCEPTS AND RATIONALE FOR ADA FEATURES.

BREAK FOR LUNCH.

ALLOW 30 MINUTES FOR THIS SECTION.

Section 4

Ada Through Example

INSTRUCTOR NOTES

VG 732.1

4-1i

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

TOPIC OUTLINE

BACKGROUND AND RATIONALE FOR ADA

WRITING AN ADA PROGRAM FROM BEGIN TO END

SUMMARY OF ADA PROGRAM STRUCTURE

ADA THROUGH EXAMPLE

LARGE SYSTEM DEVELOPMENT

SUMMARY OF ADA FEATURES

FOR MORE INFORMATION

INSTRUCTOR NOTES

THE TELEPHONE DIRECTORY SYSTEM IS PRESENTED SINCE IT IS EASY FOR STUDENTS OF ALL BACKGROUNDS TO VISUALIZE HOW IT MIGHT WORK. THUS THE STUDENT CAN CONCENTRATE ON THE ADA NOT THE WORKING OF THE SYSTEM. ASSURE THE STUDENTS THAT THEY WILL BE SEEING THIS MATERIAL SEVERAL TIMES -- IT IS NOT NECESSARY TO GRASP ALL THE FINE POINTS. (LEARNING ADA IS AN ITERATIVE PROCESS. YOU LEARN SOME, TRY TO USE, LEARN SOME MORE ...)

VG 732.1

 $4-2i$ [illegible]

EXAMPLE 2

- DEVELOP A TELEPHONE DIRECTORY SYSTEM FOR YOUR ORGANIZATION
- A USER OF THE DIRECTORY SYSTEM CAN LOOK-UP NAMES, ADD NEW ENTRIES, DELETE ENTRIES, OR LEAVE THE SYSTEM
- THE DATABASE OF THE DIRECTORY SYSTEM CONTAINS THE NAMES AND CORRESPONDING TELEPHONE NUMBERS

INSTRUCTOR NOTES

THE MOTIVATION FOR THE USE OF THE FOLLOWING ADA FEATURES IS PROVIDED THROUGH THE EXAMPLE:

1. PACKAGES
2. ENUMERATION TYPES
3. COMPOSITE TYPES
4. EXCEPTION HANDLER
5. GENERIC INSTANTIATION
6. INTERACTIVE I/O
7. SUBUNITS

A RICHER SET OF DESIGN CONCEPTS

IDENTIFY THE OBJECTS OF THE SYSTEM AND THE OPERATIONS TO BE DONE

OBJECTS

USER

DATABASE ENTRIES

OPERATIONS

MAKE INQUIRIES

EXIT SYSTEM

LOOK-UP

ADD

DELETE

IN TRADITIONAL LANGUAGES WE CAN BUILD MODULES ONLY AROUND THE OPERATIONS (SUBROUTINES). IN ADA WE CAN PACKAGE TOGETHER THE OPERATIONS AND THE OBJECTS THAT ARE AFFECTED.

INSTRUCTOR NOTES

THIS ILLUSTRATES THE FUNCTIONING OF THE DIRECTORY SYSTEM AS A WHOLE.

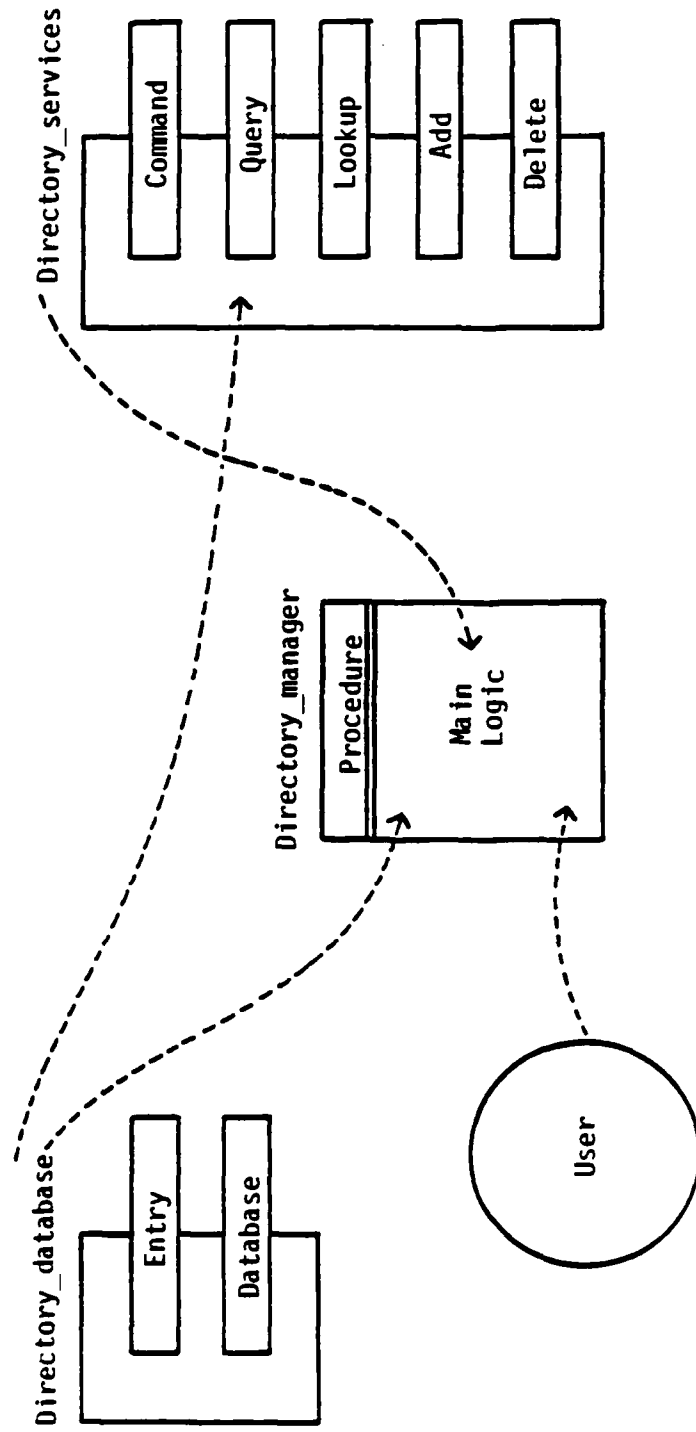
THE DOTTED LINES REFLECT THE INFORMATION DEPENDENCIES OF THE THREE PROGRAM MODULES SHOWN. THE MAIN PROCEDURE AND DIRECTORY_MANAGER NEED INFORMATION FROM THE DIRECTORY DATABASE AND THE DIRECTORY_SERVICES. THE DIRECTORY_SERVICES WILL NEED INFORMATION FROM THE DIRECTORY_DATABASE. NOTE THAT THE DIRECTORY_DATABASE DOES NOT NEED INFORMATION FROM OTHER MODULES, IT JUST PROVIDES INFORMATION TO OTHERS.

THE MAINTAINABILITY BULLET IS IMPORTANT TO EMPHASIZE. THIS BECOMES ONE OF ADA'S BEST FEATURES.

HOW THIS STRUCTURING WAS ARRIVED AT IS OUTSIDE THE SCOPE OF THIS MODULE. THIS MODULE IS LIMITED TO THE DISCUSSION OF PACKAGES WHICH GROUP RELATED DATA STRUCTURES AND/OR OPERATIONS.

IF POSSIBLE, KEEP THIS SLIDE ON SECOND OVERHEAD WHILE GOING THROUGH THE NEXT THREE SLIDES

DEVELOP A PICTORIAL REPRESENTATION OF THE SYSTEM STRUCTURE AND ISOLATE THE INTERFACES



WOULD BE EASY TO ADD/DELETE A SERVICE FOR OUR USERS. WOULD ALSO BE ABLE TO CHANGE THE
DATABASE FORMAT WITHOUT DISTURBING OUR SYSTEM = MAINTAINABILITY.

INSTRUCTOR NOTES

THE TRANSLATION OF OUR PICTURE INTO ADA CODE. FIRST, A CONCEPTUALLY PICTURE OF THE CONTENTS OF THE PACKAGE SPECIFICATION AND MAIN PROCEDURE LOGIC ARE PRESENTED FOLLOWED BY THE ACTUAL ADA CODE.

EXPRESS THE DESIGN AS ADA SPECIFICATION

```
package Directory_Database is
-- DESCRIBE WHAT TYPE OF OBJECT EACH ENTRY (NAME, TELEPHONE
-- NUMBER) IS COMPOSED OF
-- DESCRIBE WHAT THE DATABASE LOOKS LIKE
-- THIS INFORMATION CAN BE USED BY THE
-- PROCEDURE Directory_Manager AND THE
-- PACKAGE Directory_Services
end Directory_Database;
```

SPECIFICATION:

VISIBLE
PART

INSTRUCTOR NOTES

THE "with" CONTEXT CLAUSE ALLOWS THE INFORMATION FROM THE PACKAGE LISTED IN THE CLAUSE TO BE REFERENCED BY THE PACKAGE BEING DECLARED. (THIS IS ONLY PARTIALLY TRUE, BUT GETS THE CONCEPT ACROSS.) THIS IS OUR DOTTED LINES ON THE SYSTEM STRUCTURE PICTURE.

INSTRUCTOR SHOULD UNDERLINE RESERVED WORDS. ALSO NOTE THE CODING CONVENTION FOR NAMES.

```

with Directory_Database;      -- OUR INTERFACES

package Directory_Services is

    -- DESCRIBE WHAT OUR COMMAND OBJECT IS

    -- DESCRIBE WHAT OPERATIONS THE USER CAN PERFORM
    -- ON OUR DIRECTORY DATABASE
    -- OUR OPERATIONS ARE QUERY, LOOK-UP, ADD,
    -- AND DELETE
    -- THESE WILL BE EXPRESSED AS SUBPROGRAMS

end Directory_Services;

```

INSTRUCTOR NOTES

```

with Directory_Database, Directory_Services;    -- OUR INTERFACES

procedure Directory_Manager is
begin -- Directory_Manager
    -- THIS IS THE MAIN LOGIC OF OUR SYSTEM
    -- TO OUR DIRECTORY USER
    -- DEPENDING ON THE SERVICE REQUESTED BY OUR
    -- USER, ONE OF THE DIRECTORY SERVICES WOULD
    -- BE PERFORMED OR THE USER CAN EXIT THE
    -- DIRECTORY SYSTEM

end Directory_Manager;

```

INSTRUCTOR NOTES

DON'T EXPLAIN WHAT THESE ARE. JUST TELL THE STUDENT THAT THEY WILL BE DISCUSSED IN
CONTEXT.

AS WE COMPLETE OUR ADA SYSTEM WE WILL SEE THE USE OF THE FOLLOWING ADA FEATURES

- TYPES AND DECLARATIONS
- CONTROL STRUCTURES/STATEMENTS
- SUBPROGRAMS
- PACKAGES
- EXCEPTION HANDLERS
- INSTANTIATION OF GENERICS
- INTERACTIVE I/O

INSTRUCTOR NOTES

REITERATE WHAT THIS PACKAGE DOES AND WHAT WE WERE GOING TO PUT IN THIS SPECIFICATION.
THEN DISCUSS HOW THAT IS DONE IN THE SLIDE. POINT OUT WHAT THE TYPES ARE, HOW THEY FIT
TOGETHER AS A UNIT, WHAT THE PACKAGE CONCEPT DOES, ETC.

"NOW PULLING ALL THIS TOGETHER, WE HAVE..."

```

package Directory_Database is

type Index is range 1 .. 3000;           --AN INTEGER TYPE
type Name_Type is
  record
    First      : String (1 .. 10);
    Middle_Initial : String (1 .. 1);
    Last       : String (1 .. 20);
  end record;

type Telephone_Number_Type is range 1000 .. 4999;

type Directory_Unit_Record is
  record
    Name           : Name_Type;
    Telephone_Number : Telephone_Number_Type;
  end record;

type Database_Type is array (Index) of Directory_Unit_Record;

end Directory_Database;
--AN ARRAY TYPE

```

INSTRUCTOR NOTES

SAME PROCEDURE AS PREVIOUS SLIDE.

--WE WILL USE WHAT WAS IN THE PACKAGE

with Directory_Database;

package Directory_Services is

Database : Directory_Database.Database_Type;

-- OBJECT DECLARATION.

type Command is (Lookup, Add, Delete, Quit);

--AN ENUMERATION TYPE TO REFLECT
--IN OUR PROGRAMMING LANGUAGE THE
--REAL WORLD SITUATION

function Query return Command;

procedure Lookup_Entry (Data_Name : in Directory_Database.Name_Type);

procedure Add_To_Database (Data_Record :

in Directory_Database.Directory_Unit_Record);

procedure Delete_From_Database (Data_Name : in Directory_Database.Name_Type);

procedure Load_Database;

procedure Store_Database;

end Directory_Services;


```

with Directory_Database, Directory_Services;
use Directory_Services;

--WILL LET US USE A SHORT HAND NOTATION WHEN
--WE REFERENCE THE SUBPROGRAM IN THIS PACKAGE

procedure Directory_Manager is

    Local_Data_Record: Directory_Database.Directory_Unit_Record;

    procedure Input_Data (Data_Record :
        out Directory_Database.Directory_Unit_Record) is separate; --WE WOULD FIND
        --THE CODE SOME PLACE ELSE
        --THIS ALLOWS FOR SEPARATE DEVELOPMENT AND
        --COMPILATION

begin -- Directory_Manager
    Load_Database;
    loop

        case Query is

            --A CASE STATEMENT PROVIDES MULTIPLE STATES
            --TO BE HANDLED IN ONE STATEMENT

            when Lookup => Input_Data (Local_Data_Record);
                           Lookup_Entry (Local_Data_Record.Name);

            when Add      => Input_Data (Local_Data_Record);
                           Add_To_Database (Local_Data_Record);

            when Delete => Input_Data (Local_Data_Record);
                           Delete_From_Database (Local_Data_Record.Name);

            when Quit    => exit;                                -- EXIT FROM LOOP

            end case;

            end loop;
            Store_Database;
            end Directory_Manager;

```

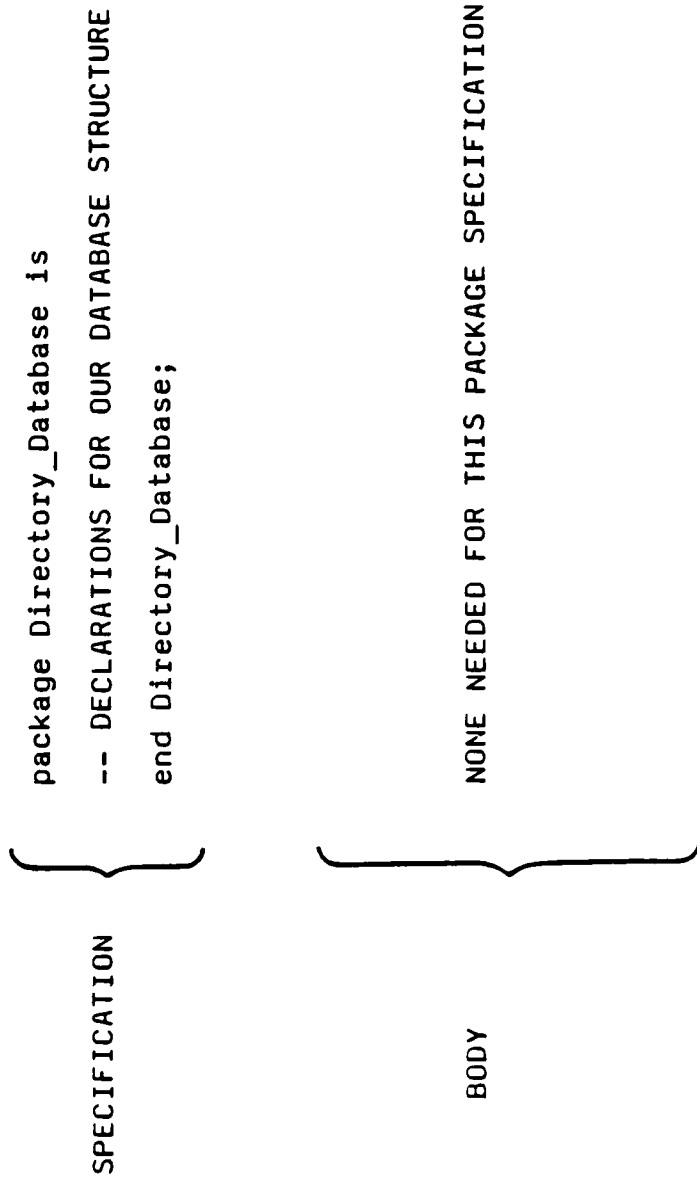
INSTRUCTOR NOTES

A SPECIFICATION USUALLY HAS A BODY, SO WE COMPLETE OUR SYSTEM FURTHER.

IN SKELETAL FORM OUR SPECIFICATION AND BODY FOR THE DATABASE PACKAGE.

REMIND THE STUDENTS WHAT THE SPECIFICATION AND BODY EACH DO.

IMPLEMENT THE PACKAGE BODIES



INSTRUCTOR NOTES

TO SHOW THE STRUCTURE OF THE PACKAGE BODY ... NESTING OF OTHER PROGRAM UNITS.

VG 732.1

4-131

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

```

with Directory_Database;
package Directory_Services is

    -- SPECIFICATIONS OF SERVICE ROUTINES

end Directory_Services;

with Text_IO;
package body Directory_Services is

    -- ANY ADDITIONAL DATA OR SUBPROGRAM DECLARATIONS
    -- NEEDED FOR THE IMPLEMENTATION

    procedure Lookup_Entry (Data_Name : in Directory_Database.Name_Type) is separate;

    -----

    function Query return Command is separate;

    procedure Add_To_Database (Data_Record:

                                in Directory_Database.Directory_Unit_Record) is separate;

    procedure Delete_From_Database (Data_Name: in Directory_Database.Name_Type) is separate;

    procedure Load_Database is separate;

    procedure Store_Database is separate;

end Directory_Services;

```

INSTRUCTOR NOTES

TO ILLUSTRATE SOME "EXECUTABLE" ADA CODE FOR ONE PROCEDURE ONLY.

```

separate (Directory_Services)  --THE PARENT UNIT
procedure Lookup_Entry (Data_Name : in Directory_Database.Name_Type) is

    Found : Boolean := False;  --INITIALIZATION AND OBJECT DECLARATION CAN BE COMBINED
    package Int_IO is new Text_IO.Integer_IO (Directory_Database.Telephone_Number_Type);
    use Int_IO;
    use Directory_Database;

begin  --Lookup_Entry

    for I in Database'Range loop

        if Database(I).Name = Data_Name then

            Text_IO.Put ("Telephone number is");  --THIS WRITES A MESSAGE TO OUR USER
            Int_IO.Put (Database(I).Telephone_Number);

            -- FOLLOWED BY THE REQUESTED PHONE
            -- NUMBER

            Found := True;

            exit;  -- EXIT THE FOR LOOP

        end if;

    end loop;

    if not Found then

        Text_IO.Put ("Name not found");

        end if;

    end Lookup_Entry;

```

INSTRUCTOR NOTES

FUNCTION QUERY...

THIS FUNCTION WILL PERFORM OUR QUERY WITH THE USER. NOTE THE USE OF ENUMERATION TYPE COMMAND, THE GENERIC INSTANTIATION FOR I/O (TELL THEM WHY THIS IS DESIRABLE - I.E. USER HAS COMPLETE CONTROL OVER I/O), THE EXCEPTION HANDLER, AND USER-FRIENDLY INTERACTION.

```

separate (Directory_Services)
function Query_return_Command is
    Query_State: Command;
package Command_IO is new Text_IO.Enumeration_IO (Command);
--GENERIC INSTANTIATION GIVES OUR PROGRAM
--A COPY OF THE I/O PACKAGE NEEDED FOR ONE
--ENUMERATION TYPE

    use Command_IO;

begin -- Query

    loop    --LET'S US PROVIDE FOR INPUT COMMAND ERRORS SO USER CAN HAVE
            --MULTIPLE CHANCES TO ENTER A VALID COMMAND WITHOUT CRASHING THE SYSTEM

        begin

            Text_IO.New_Line;

            Text_IO.Put ("Enter Command (Lookup, Add, Delete, Quit):");

            Get (Query_State);

            return Query_State;

        exception    --EXCEPTION HANDLER FOR INPUT COMMAND ERRORS

            when Text_IO.Data_Error => Text_IO.Put ("Invalid Command, Try again.");

        end;

    end loop;

end Query;

```


Section 5

Large System Development

INSTRUCTOR NOTES

TOPIC OUTLINE

BACKGROUND AND RATIONALE FOR ADA

WRITING AN ADA PROGRAM FROM BEGIN TO END

SUMMARY OF ADA PROGRAM STRUCTURE

ADA THROUGH EXAMPLE

LARGE SYSTEM DEVELOPMENT

SUMMARY OF ADA FEATURES

FOR MORE INFORMATION

INSTRUCTOR NOTES

UNDERLYING ALL LARGE SYSTEMS DEVELOPMENT REQUIREMENTS IS THE ABILITY TO HAVE MANY PROGRAMMERS WORKING SIMULTANEOUSLY. TO HAVE THAT YOU NEED THE FOUR SUB-POINTS.

DEVELOPMENT OF LARGE SYSTEMS

- PROGRAMMERS NEED TO WORK CONCURRENTLY

- RICH VARIETY OF MODULES
- IMPLEMENTATION STRATEGIES
- SEPARATE COMPILATION
- NAME SPACE CONTROL

DESIGN MODULES

- In FORTRAN,

A MODULE $\approx$ A SUBROUTINE

- In ADA,

A MODULE $\approx$ A PROGRAM UNIT

INSTRUCTOR NOTES

ADA PROGRAM UNITS

- ADA PROVIDES GREATER VARIETY OF REPRESENTATION OF DESIGN MODULES

- PROCEDURE/FUNCTIONS (ALGORITHMS)

- PACKAGES (ABSTRACT DATA TYPES)

- TASKS (PARALLEL ACTIONS)

- GENERICS (REUSABLE COMPONENTS)

- THIS ALLOWS ADA TO SUPPORT DIFFERENT DESIGN STYLES

- DATA FLOW ORIENTED

- DATA STRUCTURE ORIENTED

- OBJECT-ORIENTED

INSTRUCTOR NOTES

REMIND THE STUDENTS WHAT "LIBRARY UNITS" ARE.

BOTTOM-UP IMPLEMENTATION

THROUGH REUSABLE LIBRARY UNITS

```
package Directory_Database is
```

```
-- THE LIBRARY UNIT
```

```
end Directory_Database;
```

```
with Directory_Database, Directory_Services;  
procedure Directory_Manager is
```

```
-- "CONTEXT SPECIFICATION" ALLOWS  
-- SERVICES (OPERATIONS) AVAILABLE  
-- IN THE LIBRARY UNIT TO BE USED  
-- BY THIS PROGRAM UNIT
```

```
...
```

```
end Directory_Manager;
```

INSTRUCTOR NOTES

INDICATE THE STUBS.

NOTE ALSO, NO SEMICOLON AFTER separate (Directory_Services).

TOP-DOWN IMPLEMENTATION

THROUGH SUBUNITS (STUBS)

```
with Text_IO;  
package body Directory_Services is  
  ...  
  procedure Lookup_Entry (Data_Name:  
    in Directory_Database.Name_Type) is separate;  
  ...  
end Directory_Services;
```

PARENT UNIT

```
separate (Directory_Services)  
procedure Lookup_Entry (Data_Name:  
  in Directory_Database.Name_Type is
```

SUBUNIT

```
  ...  
begin -- Lookup_Entry  
  ...  
end Lookup_Entry;
```

INSTRUCTOR NOTES

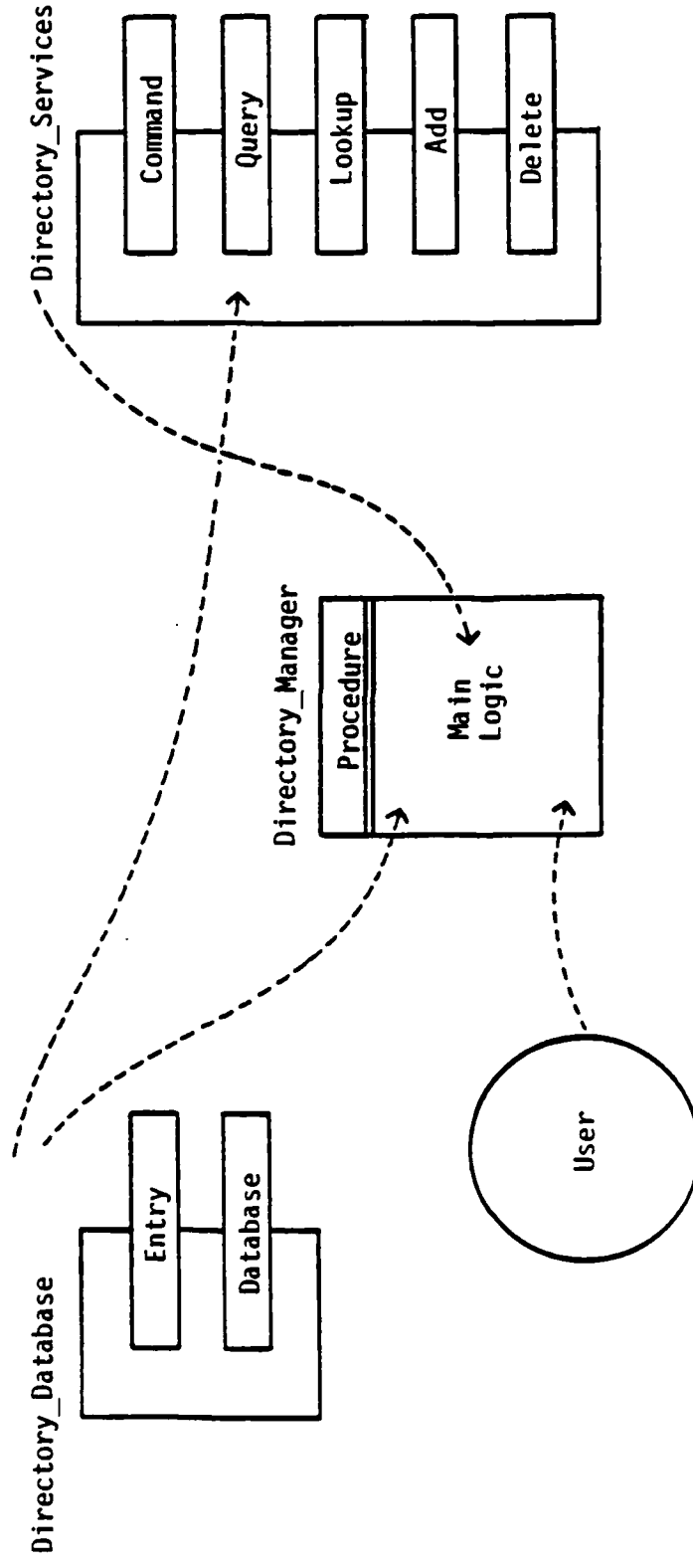
VG 732.1

5-7i

1
 2
 3
 4
 5
 6
 7
 8
 9
 10
 11
 12
 13
 14
 15
 16
 17
 18
 19
 20
 21
 22
 23
 24
 25
 26
 27
 28
 29
 30
 31
 32
 33
 34
 35
 36
 37
 38
 39
 40
 41
 42
 43
 44
 45
 46
 47
 48
 49
 50
 51
 52
 53
 54
 55
 56
 57
 58
 59
 60
 61
 62
 63
 64
 65
 66
 67
 68
 69
 70
 71
 72
 73
 74
 75
 76
 77
 78
 79
 80
 81
 82
 83
 84
 85
 86
 87
 88
 89
 90
 91
 92
 93
 94
 95
 96
 97
 98
 99
 100
 101
 102
 103
 104
 105
 106
 107
 108
 109
 110
 111
 112
 113
 114
 115
 116
 117
 118
 119
 120
 121
 122
 123
 124
 125
 126
 127
 128
 129
 130
 131
 132
 133
 134
 135
 136
 137
 138
 139
 140
 141
 142
 143
 144
 145
 146
 147
 148
 149
 150
 151
 152
 153
 154
 155
 156
 157
 158
 159
 160
 161
 162
 163
 164
 165
 166
 167
 168
 169
 170
 171
 172
 173
 174
 175
 176
 177
 178
 179
 180
 181
 182
 183
 184
 185
 186
 187
 188
 189
 190
 191
 192
 193
 194
 195
 196
 197
 198
 199
 200
 201
 202
 203
 204
 205
 206
 207
 208
 209
 210
 211
 212
 213
 214
 215
 216
 217
 218
 219
 220
 221
 222
 223
 224
 225
 226
 227
 228
 229
 230
 231
 232
 233
 234
 235
 236
 237
 238
 239
 240
 241
 242
 243
 244
 245
 246
 247
 248
 249
 250
 251
 252
 253
 254
 255
 256
 257
 258
 259
 260
 261
 262
 263
 264
 265
 266
 267
 268
 269
 270
 271
 272
 273
 274
 275
 276
 277
 278
 279
 280
 281
 282
 283
 284
 285
 286
 287
 288
 289
 290
 291
 292
 293
 294
 295
 296
 297
 298
 299
 300
 301
 302
 303
 304
 305
 306
 307
 308
 309
 310
 311
 312
 313
 314
 315
 316
 317
 318
 319
 320
 321
 322
 323
 324
 325
 326
 327
 328
 329
 330
 331
 332
 333
 334
 335
 336
 337
 338
 339
 340
 341
 342
 343
 344
 345
 346
 347
 348
 349
 350
 351
 352
 353
 354
 355
 356
 357
 358
 359
 360
 361
 362
 363
 364
 365
 366
 367
 368
 369
 370
 371
 372
 373
 374
 375
 376
 377
 378
 379
 380
 381
 382
 383
 384
 385
 386
 387
 388
 389
 390
 391
 392
 393
 394
 395
 396
 397
 398
 399
 400
 401
 402
 403
 404
 405
 406
 407
 408
 409
 410
 411
 412
 413
 414
 415
 416
 417
 418
 419
 420
 421
 422
 423
 424
 425
 426
 427
 428
 429
 430
 431
 432
 433
 434
 435
 436
 437
 438
 439
 440
 441
 442
 443
 444
 445
 446
 447
 448
 449
 450
 451
 452
 453
 454
 455
 456
 457
 458
 459
 460
 461
 462
 463
 464
 465
 466
 467
 468
 469
 470
 471
 472
 473
 474
 475
 476
 477
 478
 479
 480
 481
 482
 483
 484
 485
 486
 487
 488
 489
 490
 491
 492
 493
 494
 495
 496
 497
 498
 499
 500
 501
 502
 503
 504
 505
 506
 507
 508
 509
 510
 511
 512
 513
 514
 515
 516
 517
 518
 519
 520
 521
 522
 523
 524
 525

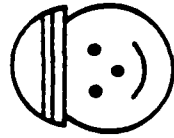
PURPOSE OF SEPARATE COMPILATION

- ALLOWS SEVERAL PEOPLE TO IMPLEMENT A SYSTEM,
FOR EXAMPLE:

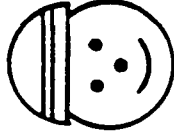
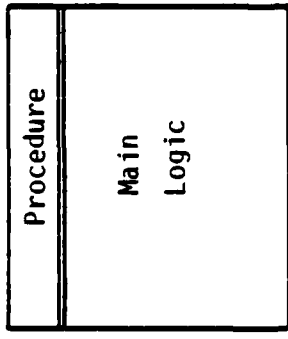


INSTRUCTOR NOTES

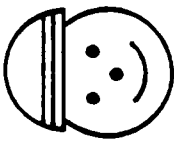
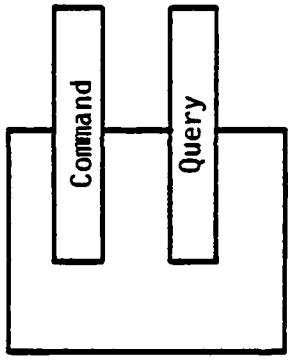
SEPARATE COMPILATION (Continued)



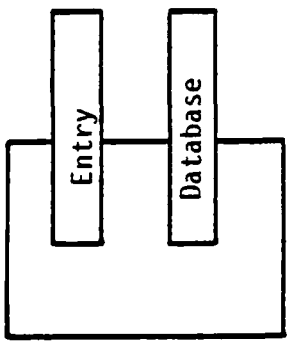
Directory_Manager



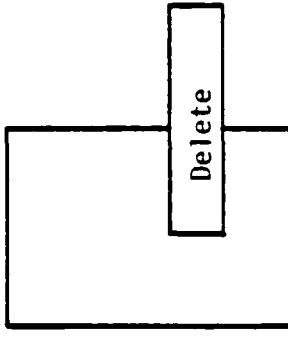
Directory_Services



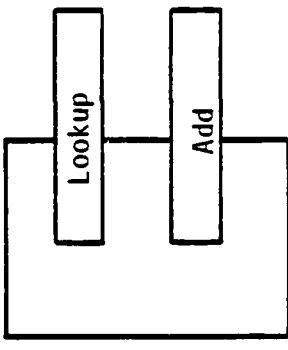
Directory_Database



Directory_Services



Directory_Services



INSTRUCTOR NOTES

CONTROL OVER ENTITY NAMES

- SCOPE/VISIBILITY RULES
- OVERLOADING

INSTRUCTOR NOTES

VG 732.1

5-10i

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000

PURPOSE OF SCOPE AND VISIBILITY RULES

- SCOPE RULES CONTROL THE LIFE TIME OF ENTITIES

FOR EXAMPLE:

WHEN STORAGE CAN BE RECLAIMED

- VISIBILITY RULES PREVENT ACCIDENTAL NAME CONFLICTS

FOR EXAMPLE:

DIFFERENT SUBPROGRAMS CAN HAVE "LOCAL" VARIABLES NAMED TEMP

INSTRUCTOR NOTES

VG 732.1

5-111i

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000

PURPOSE OF OVERLOADING

COMMON NAMES TO REFLECT SIMILAR FUNCTIONS

Put ("Name not found");

Put (Command);

INSTRUCTOR NOTES

THIS SECTION PRESENTS THE DESIGN CRITERIA FOR THE ADA LANGUAGE AND A GENERAL OVERVIEW OF THE FEATURES AND CONSTRUCTS THAT MAKE UP THE LANGUAGE. PROVIDES A "FEEL" FOR THE SCOPE OF THE FEATURES AVAILABLE IN THE LANGUAGE.

ALLOW 60 MINUTES FOR THIS SECTION.

BREAK HERE FOR 15 MINUTES.

Section 6

Summary of Ada Features

INSTRUCTOR NOTES

VG 732.1

6-1i

J. 2 200 120 222 244 261 280 299 318 337 356 375 394 413 432 451 470 489 508 527 546 565 584 603 622 641 660 679 698 717 736 755 774 793 812 831 850 869 888 907 926 945 964 983 1002

TOPIC OUTLINE

BACKGROUND AND RATIONALE FOR ADA

WRITING AN ADA PROGRAM FROM BEGIN TO END

SUMMARY OF ADA PROGRAM STRUCTURE

ADA THROUGH EXAMPLE

LARGE SYSTEM DEVELOPMENT

SUMMARY OF ADA FEATURES

FOR MORE INFORMATION

INSTRUCTOR NOTES

THE FIRST THREE LANGUAGE REQUIREMENTS FROM THE STEELMAN DOCUMENT ARE GIVEN. (OTHERS ARE EFFICIENCY, SIMPLICITY, IMPLEMENTATION. THESE LAST THREE COULD BE QUITE CONTROVERSIAL AS TO WHETHER ADA ACTUALLY SATISFIES ITS OWN REQUIREMENTS.)

LIST IS IN ORDER OF IMPORTANCE OF DESIGN CRITERIA. SHOULD NOTE THAT RELIABILITY IS MORE IMPORTANT THAN EFFICIENCY. ALSO THAT READABILITY IS MORE IMPORTANT THAN WRITABILITY - A PROGRAM IS READ MANY MORE TIMES IN ITS LIFE TIME THAN IT IS WRITTEN.

MODERN SOFTWARE ENGINEERING PRINCIPLES INCLUDE MODULARITY, ABSTRACTION, LOCALIZATION, HIDING, UNIFORMITY, COMPLETENESS, CONFIRMABILITY. INSTRUCTOR SHOULD BE FAMILIAR WITH THESE CONCEPTS. (EXCELLENT REFERENCE IF THE INSTRUCTOR NEEDS THIS BACKGROUND: "SOFTWARE ENGINEERING: PROCESS, PRINCIPLES, AND GOALS", D.T. ROSS, J.B. GOODENOUGH, C.A. IRVINE, COMPUTER, MAY 1975).

THE ADA LANGUAGE WAS DESIGNED FOR

- GENERALITY

MEETS A WIDE SPECTRUM OF NEEDS

- RELIABILITY

PROVIDES COMPILE-TIME DETECTION OF CODING ERRORS

ENCOURAGES MODERN SOFTWARE ENGINEERING PRINCIPLES

- MAINTAINABILITY

READABILITY IS MORE IMPORTANT THAN WRITABILITY

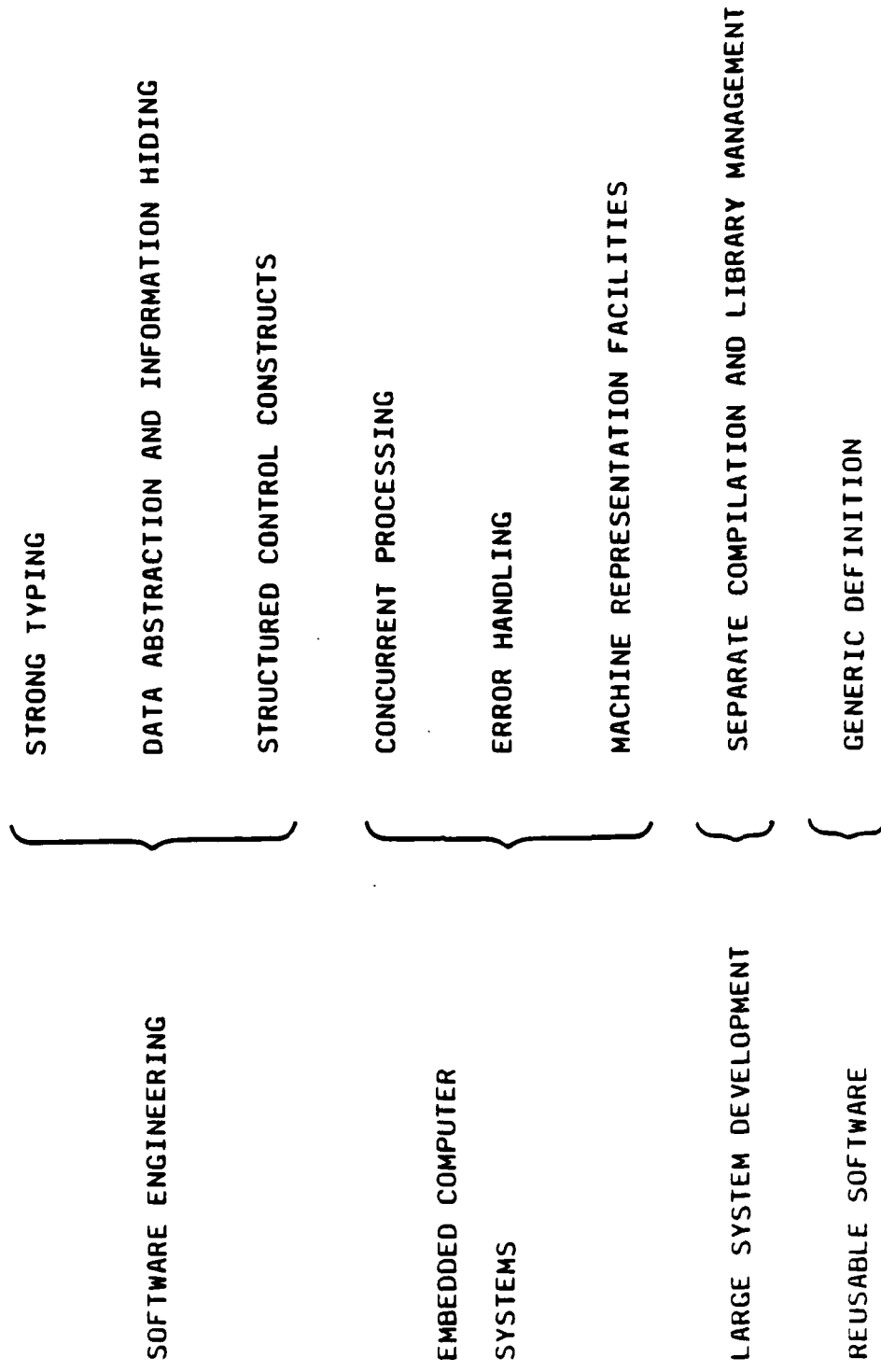
ENCOURAGES DOCUMENTATION

- MACHINE INDEPENDENCE

IMPLEMENTATION DEPENDENT LANGUAGE FEATURES CLEARLY

IDENTIFIED

DoD LANGUAGE REQUIREMENTS



37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010 1011 1012 1013 1014 1015 1016 1017 1018 1019 1020 1021 1022 1023 1024 1025 1026 1027 1028 1029 1030 1031 1032 1033 1034 1035 1036 1037 1038 1039 1040 1041 1042 1043 1044 1045 1046 1047 1048 1049 1050 1051 1052 1053 1054 1055 1056 1057 1058 1059 1060

THE LISTED ADA FEATURES WILL BE DISCUSSED. FOR EACH A DEFINITION AND ITS IMPORTANCE TO OUR SOFTWARE OBJECTIVES IS PRESENTED. THE APPROACH IS TOP-DOWN. "ADDITIONAL FEATURES" COMPRISES OTHER ADA FEATURES SUCH AS GENERICS, OVERLOADING, MACHINE REPRESENTATION SPECIFICATIONS, AND I/O.

VG 732.1

CATALOGUE OF ADA FEATURES

- PACKAGES
- SUBPROGRAMS
- TASKS
- STATEMENTS
- DECLARATIONS
- TYPES
- LEXICAL RULES
- GENERICS
- OVERLOADING
- EXCEPTIONS
- MACHINE REPRESENTATION SPECS
- I/O

INSTRUCTOR NOTES

THIS IS ONE OF ADA'S STRONGEST FEATURES.

PACKAGES PROVIDE A MEANS TO PHYSICALLY GROUP LOGICALLY RELATED OBJECTS AND OPERATIONS IN SUCH A WAY THAT WHEN WE NEED TO CHANGE PORTIONS OF A SYSTEM WE CAN KNOW THE EXACT AREAS THAT WILL BE AFFECTED. THUS WE CAN REDUCE THE AFFECTED AREA TO A MINIMUM. THIS ALLOWS US CONTROL OF THE PROVERBIAL "RIPPLE EFFECT" ASSOCIATED WITH SYSTEM CHANGES.

PACKAGES

- ARE BASIC STRUCTURING UNITS
- GROUP FUNCTIONALLY RELATED DATA AND PROGRAM UNITS
(ENCAPSULATION)
- ARE STRUCTURE REPRESENTATIONS, NOT ALGORITHMS
- PROVIDE FOR REUSABLE SOFTWARE COMPONENTS
- INCREASE MAINTAINABILITY BECAUSE EFFECT OF CHANGES CAN
BE LOCALIZED

SUBPROGRAMS

- BASIC EXECUTABLE PROGRAM UNITS

- TWO FORMS OF SUBPROGRAMS

PROCEDURE

- CALLED BY A STATEMENT

FUNCTION

- CALLED IN AN EXPRESSION, ALWAYS RETURNS 1 RESULT

- SUBPROGRAM PARAMETERS PASS VALUES

INSTRUCTOR NOTES

TASKS PROVIDE EXPRESSION OF REAL TIME PROCESSING IN A HIGH ORDER LANGUAGE (HOL).

RENDEZVOUS PROVIDES SYNCHRONIZATION AND THE EXCHANGE OF DATA.

TASKS

- PARALLEL THREADS OF CONTROL
- CONCURRENCY REAL WITH MULTIPROCESSORS;
CONCURRENCY APPARENT WITH SINGLE PROCESSOR
- MECHANISM FOR SYNCHRONIZATION AND DATA TRANSMISSION IS
CALLED "RENDEZVOUS"
- DIRECT MAPPING OF REAL TIME PROCESSING DESIGNS INTO
THE LANGUAGE

INSTRUCTOR NOTES

THIS IS A SAMPLE TASK PROGRAM UNIT. DON'T GO INTO ANY DETAIL, JUST INDICATE ITS
SIMILARITIES TO A PROCEDURE.

TASKS

```
task Card_Reader is
  entry Get (C: out Card);
end Card_Reader;

task body Card_Reader is
  Latest_Card: Card;
begin -- Card_Reader
  loop
    Text_IO.Get (Latest_Card);
  accept Get (C: out Card) do
    C := Latest_Card;
  end Get;
  end loop;
end Card_Reader;
```

INSTRUCTOR NOTES

DON'T GO INTO THE INDIVIDUAL LISTS OF STATEMENTS. JUST SHOW THAT STATEMENTS EXIST TO HANDLE THE LISTED AREAS OF ACTION AND CONTROL (I.E. FLOW CONTROL, BASIC ACTIONS, REAL TIME ACTIONS, EXCEPTIONS).

NOTE THAT THIS IS ALL THE STATEMENTS THERE ARE TO LEARN IN ADA AND THE STATEMENTS ARE SIMILAR TO OTHER LANGUAGES.

STATEMENTS

```
if Largest_Value < List (Index) then
    Largest_Value := List (Index);
end if;
```

• PROVIDE LOGIC CONTROL OR SPECIFIC ACTIONS

FLOW CONTROL:

- GOTO
- IF (CONDITIONAL)
- CASE (CONDITIONAL)
- LOOP & EXIT (ITERATIVE)
- RETURN
- EXCEPTION HANDLERS

BASIC ACTIONS:

- SUBPROGRAM CALLS
- EXPRESSIONS
- ASSIGNMENT
- RAISE (EXCEPTIONS)

REAL TIME ACTION:

- ENTRY CALL
- ACCEPT
- ABORT
- DELAY
- SELECT

EXCEPTIONS:

- RAISE

DECLARATION SCOPE:

- BLOCK

OBJECT DECLARATIONS

Largest_Value: Scores_Type;

- ASSOCIATE A NAME WITH AN OBJECT
- ALL OBJECTS MUST BE EXPLICITLY DECLARED
- CONSTANT OBJECTS
- VARIABLE OBJECTS
- DYNAMICALLY CREATED OBJECTS (AT RUN TIME)

CHOICE OF APPROPRIATE NAMES TO ACCURATELY REFLECT THE OBJECTS USE CAN GREATLY IMPROVE THE UNDERSTANDABILITY OF A SYSTEM AND THUS MAINTAINABILITY (THEREFORE DECREASE COSTS).

INSTRUCTOR NOTES

TYPE IS CONFUSING TO MANY PEOPLE WITH ONLY A FORTRAN OR ASSEMBLY LANGUAGE BACKGROUND. SIMPLY, A TYPE IS JUST A TEMPLATE, A DESCRIPTION OF HOW SOME OBJECT WILL BEHAVE, BUT IT DOESN'T CREATE THE OBJECT IN MEMORY. A DECLARATION THEN DOES THE ACTUAL CREATION. (NOTE THE CONNECTION OF TYPE AND DECLARATION.)

EMPHASIZE STRONG TYPING ADVANTAGES AND THE EXAMPLES (BRIEFLY). IT MAKES IT SO YOU CAN'T MIX APPLES AND ORANGES ACCIDENTALLY. IF YOU WOULD NORMALLY NOT COMBINE OBJECTS (SAY IN THE REAL WORLD) THAT LOGIC CAN BE REFLECTED IN THE LANGUAGE. (THIS IS AN IMPORTANT PART OF ADA.)

TYPES

Type List_Type is array (1 .. 15) of Scores_Type;

- A TEMPLATE TO DESCRIBE (NOT CREATE)

A SET OF VALUES

THE OPERATIONS APPLICABLE TO THOSE VALUES

- PREDEFINED AND USER-DEFINED TYPES

- STRONG TYPING ALLOWS ERROR DETECTION AT COMPILE-TIME

THE TYPE OF A VARIABLE OR PARAMETER DOES NOT CHANGE ONCE CREATED

Amount_Of_Gold: Pounds;

Amount_In_Glass: Ounces;

Amount_In_Glass := Amount_Of_Gold + 1; -- ILLEGAL

- INCREASED RELIABILITY BECAUSE LANGUAGE CAN BE USED

TO PROHIBIT OBJECTS OF DIFFERING LOGICAL TYPES FROM BEING MIXED

TO EXPLICITLY STATE DESIGN CONSTRAINTS

INSTRUCTOR NOTES

ADA IS A READABLE, SENSIBLE LANGUAGE.

POINT OUT THAT THERE ARE CODING CONVENTIONS, FOR EXAMPLE, YOU DON'T JUST RANDOMLY INDENT.

GENERAL LEXICAL RULES

- FREE FORMAT FOR READABILITY

INDENTATION TO SHOW LOGICAL NESTING
SPACES, BLANK LINES PERMITTED
NO CONTINUATION SYMBOL

- COMMENTS

-- TWO DASHES INDICATE START OF COMMENT

A COMMENT TERMINATES AT END OF LINE

- UPPER/LOWER CASE PLUS UNDERSCORE (_) USED IN NAMES FOR READABILITY

AD-A165 352

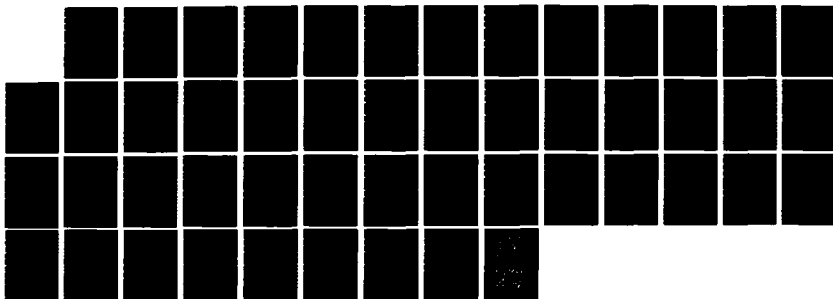
ADA (TRADEMARK) TRAINING CURRICULUM: ADA TECHNICAL
OVERVIEW L102 TEACHER'S GUIDE(U) SOFTECH INC WALTHAM MA
1986 DAB877-83-C-K586

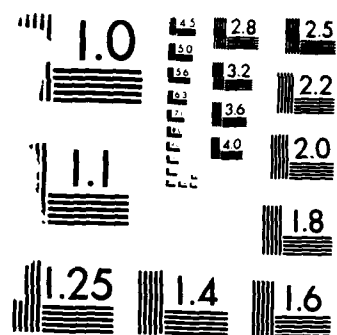
3/2

UNCLASSIFIED

F/G 5/9

NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

INSTRUCTOR NOTES

ADDITIONAL ADA FEATURES

INSTRUCTOR NOTES

GENERICS ARE SIMILAR TO MACROS BUT MACROS ARE COMPILE-TIME CONCEPTS, GENERICS ARE RUNTIME.

DIFFERENCE BETWEEN GENERICS AND SUBPROGRAMS:

SUBPROGRAMS CAN PASS ONLY VALUES AS PARAMETERS

GENERICS CAN PASS TYPES OF DATA AS WELL AS VALUES AND SUBPROGRAMS AS PARAMETERS

REUSABLE PROGRAM UNITS/SOFTWARE COMPONENTS CAN BE AN EFFECTIVE METHOD OF REDUCING OVERALL SOFTWARE COSTS ... BUT REQUIRES THOUGHT AND PLANNING.

GENERIC UNITS

- PROBLEMS THAT DIFFER ONLY IN TYPES OF DATA NEED ONLY BE SOLVED ONCE

EXAMPLE:

SORT A LIST OF NAMES

SORT A LIST OF NUMBERS

- PARAMETERIZED TEMPLATES FOR SUBPROGRAMS OR PACKAGES (NOT EXECUTABLE)
- "INSTANTIATION: CREATES AN EXECUTABLE COPY OF THE PROGRAM UNIT AND SUBSTITUTES THE PARAMETERS
- REUSABLE PROGRAM UNITS

INSTRUCTOR NOTES

GENERIC

```
generic
  Size : Positive;
  type Item is private;
package Stack is
  procedure Push (E : in Item);
  procedure Pop  (E : out Item);
  Overflow, Underflow : exception;
end Stack;

package body Stack is
  type Table is array (Positive range <>) of Item;
  Space : Table (1 .. Size);
  Index : Natural := 0;

  procedure Push(E : in Item) is
  begin
    if Index >= Size then
      raise Overflow;
    end if;
    Index := Index + 1;
    Space(Index) := E;
  end Push;

  procedure Pop(E : out Item) is
  begin
    if Index = 0 then
      raise Underflow;
    end if;
    E := Space(Index);
    Index := Index - 1;
  end Pop;
end Stack;
```

Instances of this generic package can be obtained as follows:

```
package Stack_Int is new Stack(Size => 200, Item => Integer);
package Stack_Bool is new Stack(100, Boolean);
```

INSTRUCTOR NOTES

THIS IS REALLY A FAMILIAR CONCEPT FROM OTHER LANGUAGES. WE OVERLOAD THE ADDITION OPERATOR (+) BY USING IT FOR INTEGER ADDITION AS WELL AS FOR REAL NUMBER ADDITION. WE CAN FURTHER OVERLOAD THE ADDITION OPERATOR TO ADD TWO MATRICES.

ADA JUST EXTENDS THIS POWER.

OVERLOADING

- CONCEPT OF ONE ENTITY NAME REPRESENTING TWO OR MORE ENTITIES
 Put ("Median of Scores is ");
 Put (Median);
- MOST LANGUAGES HAVE OPERATOR OVERLOADING. ADA EXTENDS THIS TO IDENTIFIER NAMES, SUBPROGRAMS, OPERATORS.
- ALLOWS PROGRAMMERS TO CHOOSE NAMES APPROPRIATE TO THEIR USE
(THE ABSTRACTION) AS LONG AS AMBIGUITY CAN BE RESOLVED BY CONTEXT

INSTRUCTOR NOTES

IN REAL TIME SYSTEMS YOU CAN'T AFFORD TO ALLOW A SYSTEM TO CRASH BECAUSE SOME "IMPOSSIBLE" STATE WAS REACHED AT SOME POINT IN THE PROGRAM. EXCEPTIONS ALLOW FOR POSSIBLE CORRECTION AND RESUMED EXECUTION, OR AT LEAST A GRACEFUL EXIT FROM EXECUTION.

NOTE:

EXCEPTIONS ARE NOT JUST FOR ERROR CONDITIONS. THEY CAN BE USED TO INDICATE WHEN SOME SPECIFIC STATE IS REACHED AND TO BRING THIS TO THE ATTENTION OF THE PROGRAM FOR HANDLING. (BACKGROUND, ONLY).

EXCEPTIONS

- AN EXCEPTION STOPS SEQUENTIAL EXECUTION WHEN A PARTICULAR CONDITION IS REACHED, AND TRANSFERS CONTROL TO SOME KNOWN LOCATION WHERE THE CONDITION MAY BE HANDLED
- A MECHANISM FOR FAULT-TOLERANT PROGRAMMING
ALTERNATIVE TO EXPLICIT ERROR CODE PARAMETERS
- PREDEFINED AND USER-DEFINED EXCEPTIONS
- AID TO RELIABILITY

INSTRUCTOR NOTES

EXCEPTION

```
begin
...
exception      --EXCEPTION HANDLER
when Division_By_Zero =>
...
when others =>
...
end;
```

INSTRUCTOR NOTES

THE LAST 2 BULLETS ARE THE MAIN IMPACT, THRUST OF THIS FACILITY.

FACILITY IS PRIMARILY NEEDED FOR ECS USE (ONLY SPECIALIZED FEW WILL NEED TO USE THIS FEATURE).

BY ENCAPSULATING THE MACHINE DEPENDENT CODE, THE SYSTEM IS EASIER TO MAINTAIN OR RETARGET BECAUSE THE AREAS OF NECESSARY CHANGE ARE LOCALIZED AND IDENTIFIED.

MACHINE REPRESENTATION SPECIFICATIONS

for Vehicle_Record'Size use 1000;

- MAPS AN OBJECT DESCRIPTION (A TYPE) ONTO ACTUAL HARDWARE
- CREATES INTERFACES WITH FEATURES OUTSIDE THE LANGUAGE (E.G. INTERRUPTS, I/O DEVICES)
- ALLOWS USER TO INTERFACE WITH HARDWARE PERIPHERALS WHILE REMAINING IN HIGH ORDER LANGUAGE
- ENCAPSULATE FOR PORTABILITY, MAINTAINABILITY

INSTRUCTOR NOTES

IF YOUR PART OF A SYSTEM HAS SPECIFIC OR LIMITED I/O NEEDS, THEN YOU ONLY HAVE TO HAVE WHAT IS ABSOLUTELY NECESSARY TO YOUR PARTICULAR FUNCTION. YOU DON'T HAVE TO HAVE ALL POSSIBLE FORMS/FORMATS OF I/O FOR ALL POSSIBLE USES. DECREASES COMPILE OVERHEAD.

6-20i

VG 732.1

INPUT/OUTPUT

- ACCESSED THROUGH PACKAGES (PREDEFINED AND USER-DEFINED)

- USER HAS COMPLETE CONTROL OF I/O

- PREDEFINED I/O

- LOW-LEVEL I/O

- HIGH-LEVEL I/O

- TEXT I/O

- DIRECT I/O

- SEQUENTIAL I/O

INSTRUCTOR NOTES

A SUMMARY OF WHAT/WHERE/WHY ADA IS USEFUL.

AGAIN, SOFTWARE ENGINEERING PRINCIPLES IMPLIES SUCH CONCEPTS AS STRUCTURED PROGRAMMING,
STRONG TYPING OF DATA, MODULARITY, ABSTRACTION, READABILITY.

EMPHASIS OF ADA

- USEFUL FOR WIDE RANGE OF APPLICATIONS
EMBEDDED COMPUTER SYSTEMS
SYSTEMS PROGRAMMING
REAL TIME PROGRAMMING
DATA PROCESSING
- DEVELOPMENT BY PROJECT TEAMS
- SOFTWARE ENGINEERING PRINCIPLES ENCOURAGED AND ENFORCED
- MAINTAINABILITY AND RELIABILITY

INSTRUCTOR NOTES

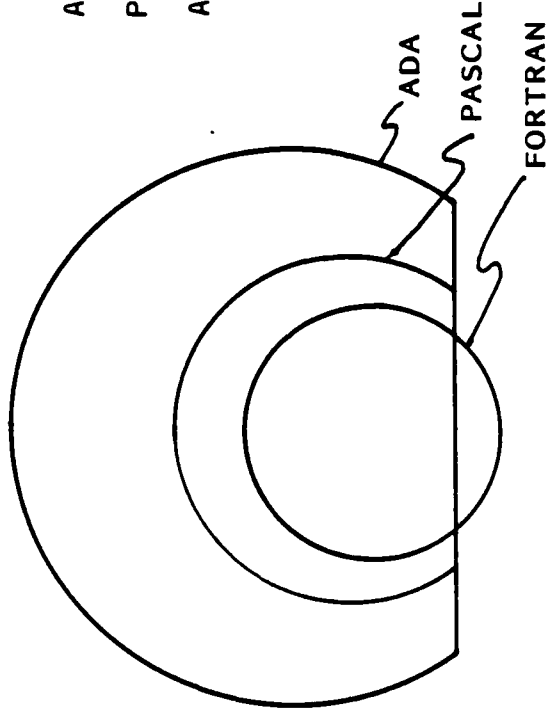
COMPARISON WITH PASCAL AND FORTRAN

ROUGHLY SPEAKING:

ADA INCLUDES ALL OF PASCAL

PASCAL INCLUDES MOST OF FORTRAN

ADA INCLUDES MOST OF FORTRAN



- I/O FORMATTING IS MORE PRIMITIVE
IN ADA AND PASCAL

INSTRUCTOR NOTES

COMPARISON WITH FORTRAN/PASCAL

ASPECTS NEW TO FORTRAN (NOT PASCAL)

MORE DATA TYPES

THE CONCEPT OF DATA TYPE

NO IMPLICIT DECLARATIONS

RICHER CONTROL STRUCTURES

INSTRUCTOR NOTES

FORTRAN COMPARISON

DIFFERENCES FROM FORTRAN (NOT PASCAL)

DATA TYPES

- PLAY A MORE CENTRAL ROLE IN ADA
- MORE TYPES
 - ENUMERATION TYPES
 - RECORD TYPES (INCLUDING VARIANTS)
 - ACCESS TYPES (POINTERS)

EXPLICIT DECLARATIONS REQUIRED

- READABILITY
- CATCHES ERRORS

MORE CONTROL STRUCTURES

- CASE STATEMENT

INSTRUCTOR NOTES

FORTRAN COMPARISON: DATA TYPES

- VALUES, E.G., Mon, Wed
- OPERATIONS, E.G. Mon < Y
Mon + Wed -- ILLEGAL
- CONSTRAINTS - RESTRICT VALUES, NOT OPERATIONS
range Mon .. Fri

CAN DEFINE NEW, PROBLEM-ORIENTED DATA TYPES IN ADA

INSTRUCTOR NOTES

VG 732.1

6-26i

FORTAN COMPARISON: DATA TYPES

ENUMERATION TYPES

VALUES

type Day is (Mon, Tue, Wed, Thu, Fri, Sat, Sun);

subtype Workday is Day range Mon .. Fri;
CONSTRAINT

Today : Day := Sat;

Y : Workday := 3;

-- ILLEGAL; NOT VALUE OF TYPE DAY

Holiday : Workday := Today;

-- EXCEPTION; CONSTRAINT NOT SATISFIED

... Mon + Today

-- ILLEGAL OPERATION

- MORE READABLE

- ERRORS ARE CAUGHT

- REQUIRES ADVANCE PLANNING TO CREATE TYPES THAT MEET YOUR NEEDS

INSTRUCTOR NOTES

VG 732.1

6-271

FORTRAN COMPARISON: RECORD TYPE

```
type Months is (Jan, Feb, Mar, Apr, May ... Dec);
```

```
type Date is
```

```
record
```

```
    Month : Months;
```

```
    Day   : Integer range 1 .. 31;
```

```
    Year  : Integer range 1800 .. 2500;
```

```
end record;
```

```
X : Date := (Nov, 5, 1981);
```

```
Y : Date := (Year => 1901, Month => Nov, Day => 5);
```

```
if Y.Year > 1940 then ...
```


FORTRAN COMPARISON: CONTROL STRUCTURES

```
if Today = Thu then          -- IN FORTRAN 77
...
else
...
end if;

case Today is                -- NOT IN FORTRAN 77
    when Mon .. Thu => Work;
    when Fri => Work; Celebrate;
    when Sat|Sun => Rest; -- when others => Rest;
end case;
```

- FULL SET NEEDED FOR STRUCTURED PROGRAMMING

INSTRUCTOR NOTES

DIFFERENCES FROM PASCAL

* PACKAGES

- DATA
- TYPES
- OPERATIONS (SUBROUTINES)
- PRIVATE TYPES (PORTABILITY; ABSTRACTION)

* SEPARATE COMPILATION (WITH INTERFACE CHECKING)

* CONCURRENT AND REALTIME PROCESSING

REPRESENTATION CONTROL - SPACE EFFICIENCY

- PACK DATA
- CONFORM TO EXTERNAL INTERFACES

LOW LEVEL -- ACCESS TO MACHINE ARCHITECTURE

- MACHINE CODE

GENERIC UNITS

- ENHANCE REUSABILITY

EXCEPTION CONDITIONS

FIXED-POINT ARITHMETIC

INSTRUCTOR NOTES

ALLOW 15 MINUTES FOR THIS SECTION.

Section 7

For More Information

INSTRUCTOR NOTES

TOPIC OUTLINE

BACKGROUND AND RATIONALE FOR ADA

WRITING AN ADA PROGRAM FROM BEGIN TO END

SUMMARY OF ADA PROGRAM STRUCTURE

ADA THROUGH EXAMPLE

LARGE SYSTEM DEVELOPMENT

SUMMARY OF ADA FEATURES

FOR MORE INFORMATION

INSTRUCTOR NOTES

ADA/JUG IS A USER-ORIENTED GROUP WITH TOP DEFENSE CONTRACTORS, IMPLEMENTORS, EDUCATORS, GOVERNMENT OFFICIALS (AF, AJPO) MEETING TO EXCHANGE CURRENT STATUS, CONCERNS, NEW IDEAS.

SIGADA HAS MORE OF AN IMPLEMENTORS, RESEARCH BENT.

ADA LETTERS IS A PUBLICATION OF THE SPECIAL ADA INTEREST GROUP OF THE ACM.

FOR MORE INFORMATION

- ADA - JOVIAL USERS GROUP (ADAJUG)
- SIGADA
- ADA LETTERS
- ADA JOINT PROGRAM OFFICE (AJPO)
- ARPANET
- SEMINARS
- BOOKS

INSTRUCTOR NOTES

ADA JUG

LANGUAGE CONTROL FACILITY

CAROLE STEELE

ASD/ADOL

WRIGHT-PATTERSON AFB, OH 45433

(513) 255-4472

INSTRUCTOR NOTES

VG 732.1

7-41

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

SIGADA AND ADA LETTERS

SIGADA: TECHNICAL STUDY GROUP OF THE ACM

ADA LETTERS: SIGADA PUBLICATION

FOR MEMBERSHIP IN ACM SIGADA

ACM, INC.

P.O. BOX 12115

CHURCH STREET STATION

N.Y., N.Y., 10249

INSTRUCTOR NOTES

VG 732.1

7-5i

AJPO

ADA JOINT PROGRAM OFFICE

1211 SOUTH FERN STREET

ROOM C-107

ARLINGTON, VA 22202

(202) 694-0208 (ADA INFORMATION CLEARINGHOUSE)

Material: Ada Technical Overview (L102)

We would appreciate your comments on this material and would like you to complete this brief questionnaire. The completed questionnaire should be forwarded to the address on the back of this page. Thank you in advance for your time and effort.

1. Your name, company or affiliation, address and phone number.

2. Was the material accurate and technically correct?

Yes ☐

No ☐

Comments:

3. Were there any typographical errors?

Yes ☐

No ☐

If yes, on what pages?

4. Was the material organized and presented appropriately for your applications?

Yes ☐

No ☐

Comments:

5. General Comments:

place
stamp
here

COMMANDER
US ARMY MATERIEL COMMAND
ATTN: AMCDE-SB (OGLESBY)
5001 EISENHOWER AVENUE
ALEXANDRIA, VIRGINIA 22233

END
DTIC
FILMED
4-86