ADA HIGH ORDER LANGUAGE TRAINING USING COMPUTER-BASED EDUCATION

David G. Stephan
Government Systems, Control Data Corporation
Minneapolis, Minnesota

Charles B. Johnston
Education, Control Data Corporation
Minneapolis, Minnesota

ABSTRACT

The Department of Defense (DOD) is spending an estimated $6 billion on software each year, and this budget is forecasted to increase by 15 to 20 percent annually. In recognition of this software cost spiral and to halt the proliferation of computer programming languages, DOD has sponsored the development of a new language--called Ada--to provide a standard, computer-independent high order language for major defense systems software. Ada is expected to have widespread application throughout DOD and will require training for thousands of individuals. Control Data is developing a solution to the Ada training problem using the Control Data PLATO® computer-based education system.

THE PROBLEM

The U.S. Department of Defense initiated Ada, the new high order language (HOL), in 1976 with the objective of eventually standardizing programming languages into one HOL. The five-year design effort involved defense departments, academia and industry from fifteen countries. Ada is currently going through American National Standards Institute procedures to become a U.S. standard and is expected to become an international standard through the International Organization for Standardization. There is widespread interest in Ada, and it is believed that a requirement for it will evolve in commercial as well as defense applications.

DOD's Software Problem

DOD is spending an estimated $6 billion on software each year and as indicated in figure 1, the software budget is forecasted to increase rapidly. (1)

In recognition of this accelerating software cost spiral, DOD established policy framework (DOD Directive 5000.29) in 1976 for advanced computer technology, including the requirement for a DOD approved HOL. The resulting development of Ada is now history, and the design of Ada culminated in the DOD "Ada Debut" on September 4, 1980.

Ada began in DOD as one of the initiatives aimed at controlling the "embedded computer" software costs. Embedded computers are those equipments procured as part of weapons systems and controlled by the DOD 5000.XX series of directives. Ada is initially being hosted on automatic data processing computers used in support of weapons systems. Furthermore, Ada is perceived as being a system design language which opens the door for training of hardware and system designers, program managers, as well as software designers.

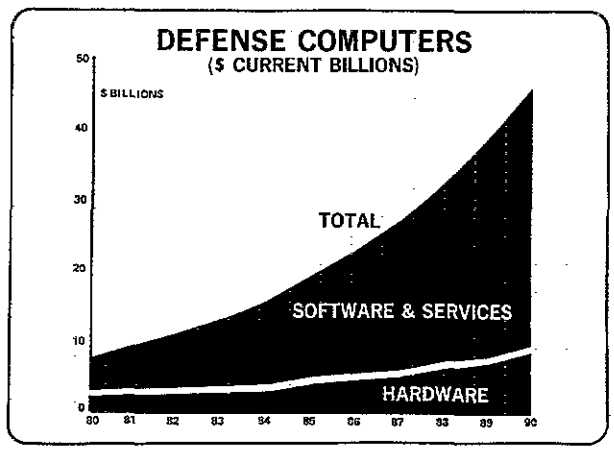


Figure 1. Software budget forecast

As indicated in figure 2, the costs of computer hardware are continuing to decrease, but software costs, being labor-intensive, are rapidly increasing. A sizable portion of software life-cycle costs can be attributed to maintenance and training.

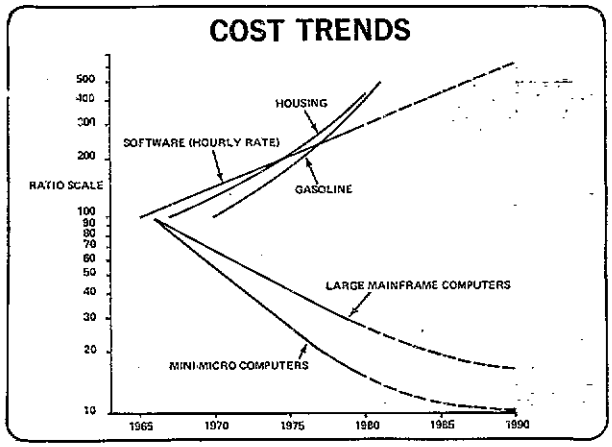


Figure 2. Hardware vs. software cost forecast

A fundamental problem confronting industry and DOD is the shortage of computer programmers. Figure 3 shows that the number of computers in the U.S. is growing much faster than the number of programmers and, unfortunately, programmer productivity is not significantly improving.

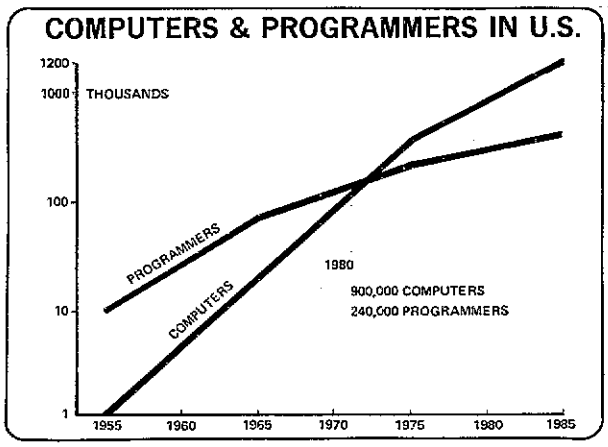


Figure 3. Growth rate of programmers vs. computers

The software labor shortfall is likely to get worse. There is an industry-wide need to recruit and train more computer people.

The Ada Training Problem

More and more computers will be used by DOD, requiring more application software. Adding this to the labor shortfall problem, and the rapidly escalating cost of software resources, makes the need for training apparent.

At least four classes of education and training are needed to successfully implement Ada throughout DOD:

1. An overview of Ada must be provided for managers and executives and as a first course for software and systems personnel.

2. A retraining course for programmers who are already knowledgeable on another HOL, such as FORTRAN, is required.

3. A complete training program from computer fundamentals through the Ada language is needed for the thousands of computer neophytes who must be trained to relieve the labor shortfall problem facing industry and DOD.

4. Unique application training on how to use the Ada HOL effectively in solving specific computer system requirements must be provided.

THE SOLUTION

Control Data is a leader in computer-based education and is currently providing hundreds of courses via the Control Data PLATO system. Control Data is planning PLATO-based courseware for Ada in three types of courses. The first will address the "whats and whys" of Ada and will culminate in an Overview of Ada course. The second course will address the "how to's" of Ada and will be an Ada Programming Fundamentals course based on the PLATO system. The third area of training will actually be a series of courses designed to provide DOD application unique training, such as "How to use Ada effectively in solving a digital signal processing problem."

These courses will satisfy the four classes of Ada training previously identified.

Individualized Instruction

Unlike traditional lecture-based instruction, where a single instructor speaks to a captive group of students, individualized instruction allows students to learn independently from sets of structured learning resources. Each learning resource is a carefully designed package of instructional materials presented by one or more types of media such as computer-assisted instruction, text, and audiovisual products (audiotapes, videotapes, filmstrips, or slide shows). These materials communicate the same knowledge presented by instructors through lectures.

Extensive research by educators and psychologists proves the importance of interaction to improve learning and strengthen retention. Multimedia individualized instruction checks students' understanding just as instructors do--by asking questions. Strategically placed questions or other response requirements elicit interactions from each student. The purpose of the interactions may be to check for understanding of concepts, require relationships to be drawn between concepts, emphasize important points, illustrate rules, or simulate procedures.

Students progress at their own rate as they work through each learning activity. Because the burden of delivering information is shifted from the instructor to the learning activities, instructors can give individual attention to each student. Instructors are freed to answer specific questions, provide encouragement, provide additional explanations to meet unique needs of individual students, and get to know each student on a personal basis. Students also receive immediate feedback on how well they are learning--they are not made to wait for quizzes given at arbitrary times whether or not they are ready.

Numerous benefits are gained from individualized instruction. Following are some of the more important benefits:

- Because students progress at their own rate, fast students are not held back and slow students do not fall behind.
- Training to the desired skill levels is more precise. Better students are not overtrained and slow students are not undertrained.
- Instructors can spend more time with students who need special assistance rather than attempting to teach to an idealized average, or slightly below average, student.
- Students are more active. They cannot sit passively while instruction goes on around them. This interaction improves learning and increases retention for each student.
- Attrition can be reduced because students are not forced to learn at an externally imposed pace they may be unable to meet. More attention can be given to specific learning problems unique to individual students; this decreases chances of failure resulting from difficulty in only one area.
- Deficiencies of the training program itself can be identified, diagnosed, and corrected quickly as a result of the concrete structure of modularized learning activities. The result is more cost-effective, efficient training.
- More precise control of training is possible because instructional materials contained in learning activities are not subject to instructors' reassignment, illness, lack of training, or forgetfulness.

In computer-managed instruction (CMI), the functions of testing, study assignments (prescriptions), and record keeping are assumed by the computer. In this application, the computer becomes a powerful tool for relieving instructors of administrative tasks. In CMI, all tests are administered on-line (on a PLATO terminal). This on-line testing enables the computer to immediately score tests, analyze item responses, and store the data in files containing complete histories of students' past performances and assignments. Decision logic tables, or algorithms, programmed into the computer can then use this pool of information to make prescriptions tailored to students' needs. These prescriptions can assign remedial work, more advanced lessons, additional tests, outside references or resources, or any one of a number of instructional alternatives desired by an instructor.

Records of learning performance stored in student data files can be examined, sorted, and arranged in any format desired by an instructor or training manager. This data may include individual scores for a given student per assignment, summary scores for all students on a given test or learning activity, number of times a particular activity is assigned, time elapsed between tests, and total hours in training.

The benefits of CMI thus lie in the precision, control, and reduced instructor burden it provides. Each assignment is constructed from a known data base. Therefore, if something is wrong with a particular assignment or test, it can be traced to the logic behind it and corrected. In traditional management modes, decisions are often based upon incomplete information or best guesses. These subjective decisions make it difficult to systematically improve a training program. A CMI program, however, can be steadily improved by building upon objective data describing each student's experience. The mass of data available can be manipulated on the computer in a matter of seconds, while it may take hours or days to do so manually. The chances of errors in transcription are also greatly reduced. This is very important when dealing with test results, where an error could mean the difference between success or failure.

In short, CMI automates most of the mechanical tasks of managing a training program. This automation not only enables instructors to spend more time with their students, it also increases control over the training environment and improves accuracy and usefulness of essential training information.

The Control Data PLATO System

The Control Data PLATO system is an interactive computer-based education system, which consists of hardware, software, a computer system, and courseware. The hardware portion consists of graphic display terminals and a communications network. The software portion runs as an application package under the control of the Network Operating System . This software supports the PLATO communications equipment, manages computer resources available at the central site, and processes all interaction between the central system and the PLATO terminal. The PLATO system runs on the CDC® CYBER 170 or CDC® CYBER 70 computer systems. Other PLATO systems, such as Micro PLATO, can be used for courseware delivery in a stand-alone mode and connected to the central PLATO system for authoring capability.

Overview of Ada

The first course, Overview of Ada, is intended to provide programmers and their managers with an understanding of the development and benefits of this new programming language. The course presents the rationale behind the design of the Ada language and a brief history of its development, as well as the economic, productivity, and reliability problems in software development. Each of the principles that were applied in an attempt to overcome the problems and produce more efficient and reliable software development is explained, along with some of the language features that were developed to achieve these principles. The purpose of this course is to introduce the Ada language so that it is viewed as a viable alternative to other programming languages.

The introduction illustrates the current economic problems faced in software development and shows why the design and development of a new language were undertaken. It highlights the history of the language, touching on key events and emphasizing the computing principles that were applied in this effort to provide greater productivity and reliability through the use of a new language.

The course explains each of these computing principles intuitively, illustrating how its application can facilitate software development. The features designed to support the principle are also presented. For example, modularity is one of the principles that was applied in the design of the Ada language. The concept of modularity is discussed, and the language feature--packages--that supports this principle is presented, along with the benefits that packages provide, such as permitting the division of labor and providing more readable code.

The course discusses other principles including reusable software, separate compilation, tasking, and strong typing. The features that were incorporated in the language to achieve these principles are also presented.

Finally, there is a discussion of the rationale for the design and development of an Ada environment. The functions that the Ada Programming Support Environment will provide and the different levels of the environment are explained.

Ada Programming Fundamentals

The second course, Ada Programming Fundamentals, is intended to provide the experienced programmer with an introduction to the Ada language. Overview of Ada is a prerequisite to this course, providing a background in the basic concepts underlying the language. The purpose of Ada Programming Fundamentals is to teach the Ada language to experienced programmers in a manner that achieves the full benefits of the new language. Teaching the syntax of the Ada language is not adequate preparation for programming with the language. Instead, many programmers need to learn

a new discipline of designing and developing programs to achieve the desired benefits from the language.

The use of a top-down modular approach in designing programs enhances the effective use of the Ada programming language. A section of this course discusses top-down modular design of Ada programs and the use of the limited set of structured control structures in designing algorithms. The remainder of the course emphasizes these approaches.

The intention of this course is to provide the experienced programmer with a working knowledge of the Ada language, so that it is possible to use it to write programs. The course is not intended to cover all variations of every language feature, but instead prepares the programmer for getting started using the language.

An Example of a PLATO Course

The addendum to this paper contains screen prints made directly from the PLATO terminal showing a lesson sequence from the Ada Programming Fundamentals course. The lesson is on the Ada "IF" statement and the screen prints show the following:

- Lesson title
- Objectives for the lesson
- Flow charts of control selection
- Explanation of "IF" statement syntax
- Example of "IF" statement
- Exercise using "IF" statement

The use of PLATO graphics greatly enhances the learning potential of the CRT, or display, terminal by providing a "picture drawing" tool for the author. Students assimilate new concepts more effectively through graphics combined with text.

References

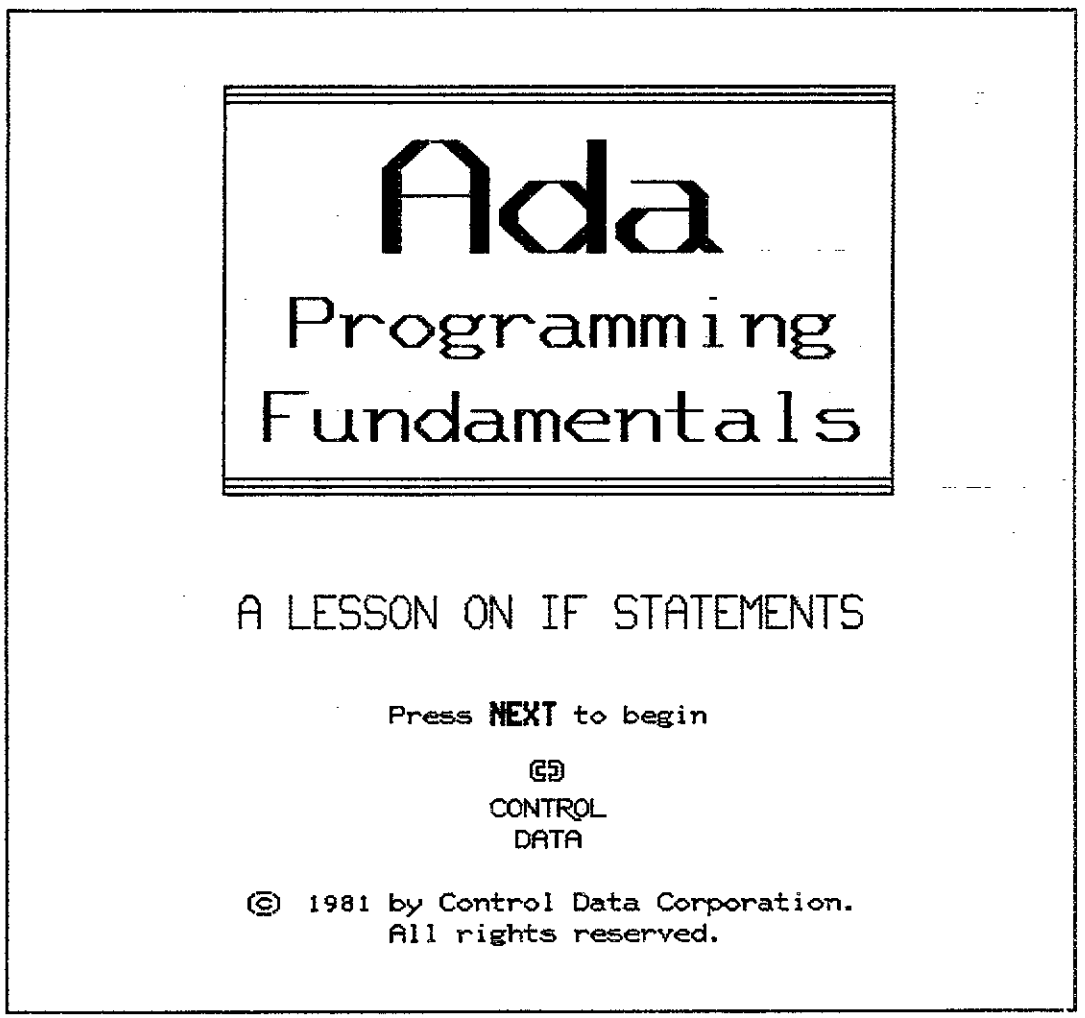
(1) Figures 1, 2, and 3 are from the DOD Digital Data Processing Study performed by an industry team chaired by Mr. David G. Stephan, Control Data Corporation, Minneapolis, Minnesota. The study was performed under the auspices of the Electronic Industries Association.

ABOUT THE AUTHORS

Mr. David G. Stephan, Manager
Planning Government Systems
Control Data Corporation.
Responsible for Control Data's Ada program.

Mr. Charles B. Johnston,
Control Data Corporation.
Responsible for designing the Ada courseware on Control Data's PLATO system.

Screen Print 1: Lesson title



Screen Print 2: Objectives for the lesson

INTRODUCTION

The purpose of this activity is to familiarize you with the if statement, its use and its syntax.

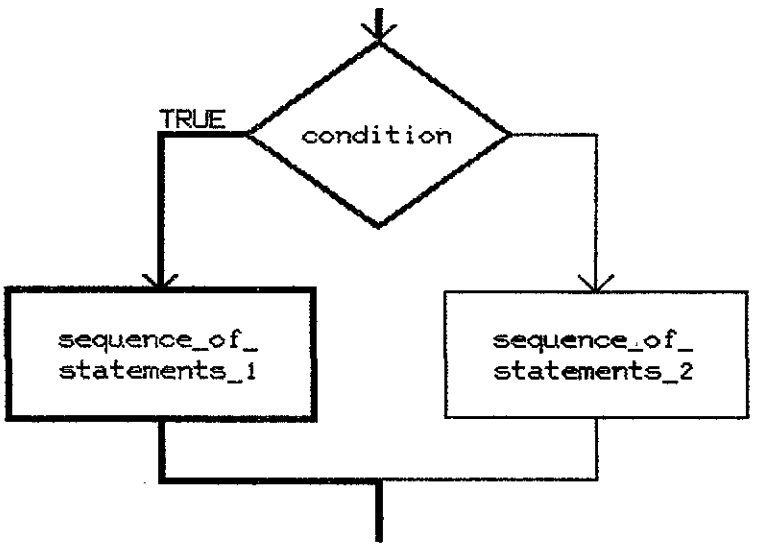
At the end of this activity you should be able to:

- Identify when an if statement should be used as a control structure.
- Code the if statement with two branches.
- Code the if statement with multiple branches.

Press **NEXT** to continue

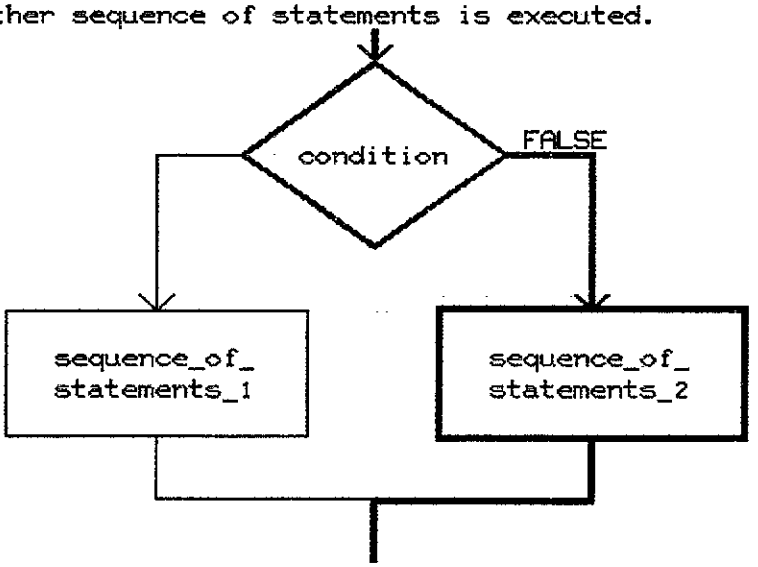
Screen Print 3, Display 1: Flow charts of control selection

The order of execution of statements in a program is generally sequential. The execution proceeds statement by statement through the program. There are circumstances where it is necessary to alter this sequential flow of the program. Conditional statements permit the choice between several courses of action depending on whether a specified condition is met. If the condition is met, one sequence of statements is executed.



Press **NEXT** to continue

The order of execution of statements in a program is generally sequential. The execution proceeds statement by statement through the program. There are circumstances where it is necessary to alter this sequential flow of the program. Conditional statements permit the choice between several courses of action depending on whether a specified condition is met. If the condition is met, one sequence of statements is executed. If the condition is not met, another sequence of statements is executed.



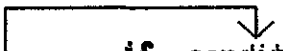
Press **NEXT** to continue

Display 2 adds this sentence.

Display 2 lightens the left side of figure and darkens the right side to correspond with new information.

Screen Print 4: Explanation of "IF" statement syntax

The syntax of the if statement is:



```
if condition then
   sequence_of_statements_1
else
   sequence_of_statements_2
end if;
```

When the condition is met, the first sequence of statements is executed, and when it is not the second sequence is executed.

The condition is any Boolean expression, such as **X <= 0.**

Press **NEXT** to continue

Screen Print 5, Display 1: Example of "IF" statement

AN EXAMPLE:

This is an example of an Ada function that uses the if statement to return the values, **TRUE** or **FALSE**, depending on whether a weight is within a given range.

```
function CHECK_WEIGHT (W:WEIGHT) return BOOLEAN is
begin
   if W in WEIGHT range 390..410
      return TRUE;
   else
      return FALSE;
   end if;
end CHECK_WEIGHT;
```

In this function, if the weight **W** is within the acceptable range, then the function returns the value **TRUE**.

Press **NEXT** to continue

AN EXAMPLE:

This is an example of an Ada function that uses the if statement to return the values, **TRUE** or **FALSE**, depending on whether a weight is within a given range.

```
function CHECK_WEIGHT (W:WEIGHT) return BOOLEAN is
begin
   if W in WEIGHT range 390..410
      return TRUE;
   else
      return FALSE;
   end if;
end CHECK_WEIGHT;
```

Display 2 removes first box and then boxes the lower information.

In this function, if the weight **W** is within the acceptable range, then the function returns the value **TRUE**.

If the weight is not within the acceptable range, then the function returns the value **FALSE**.

Display 2 adds this sentence.

Press **NEXT** to continue

Screen Print 6, Display 1: Exercise using "IF" statement

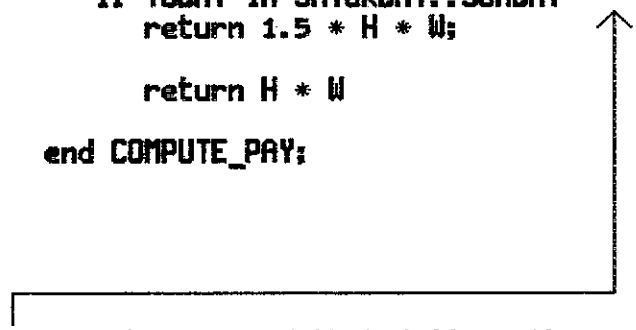
AN EXERCISE:

You will assist in writing a function that computes daily pay. If the day worked is either Saturday or Sunday, then the pay computed should be 1.5 times the normal wage.

```
function COMPUTE_PAY (H:HOURS; W:WAGE; TODAY:DAY)
   return DOLLARS is
begin
   if TODAY in SATURDAY..SUNDAY
      return 1.5 * H * W;

      return H * W

end COMPUTE_PAY;
```



Enter the keyword that follows the condition in an if statement.

Screen Print 6, Display 2

AN EXERCISE:

You will assist in writing a function that computes daily pay. If the day worked is either Saturday or Sunday, then the pay computed should be 1.5 times the normal wage.

```
function COMPUTE_PAY (H:HOURS; W:WAGE; TODAY:DAY)
   return DOLLARS is
begin
   if TODAY in SATURDAY..SUNDAY then
      return 1.5 * H * W;

      return H * W

end COMPUTE_PAY;
```

Enter the keyword that precedes the second sequence of statements in an if statement.

Upon completion of first direction, second direction appears.

AN EXERCISE:

You will assist in writing a function that computes daily pay. If the day worked is either Saturday or Sunday, then the pay computed should be 1.5 times the normal wage.

```
function COMPUTE_PAY (H:HOURS; W:WAGE; TODAY:DAY)
   return DOLLARS is
begin
   if TODAY in SATURDAY..SUNDAY then
      return 1.5 * H * W;
   else
      return H * W
   end if;
end COMPUTE_PAY;
```

Good, you have completed the coding of the if statement!

Feedback for correct answer appears.