01 Jan 1985

ADA: A NEW PROGRAMMING LANGUAGE.

Daniel C. St. Clair
Missouri University of Science and Technology

Ada: a new programming language

Daniel C. St. Clair

The Department of Defense developed an incredible new programming language and named it in honor of Ada Lovelace, the world's first programmer

Fortran and Pascal have long been used by engineers to solve problems. Now a new programming language called Ada is expected to become the language used by many engineers, especially those employed by companies who work for the U.S. Department of Defense (DOD).

The DOD is a major user of computer hardware and software systems. As you would expect, large air, ground and sea defense systems require large, complex computing systems. Many of these large systems, such as a guidance computer on a missile, are actually composed of many smaller subsystems referred to as embedded systems. Each of these embedded systems has its own set of computers and software, and may have subsystems. For the entire system to function properly the embedded systems must communicate with each other. Thus, a large system can be composed of many subsystems developed by many different companies. To further complicate matters, embedded systems may be written in Fortran, assembly language, or Pascal, resulting in large expensive systems which are difficult to implement and maintain.

In an attempt to control defense software costs while maintaining system reliability, the DOD has sponsored the development of a new programming language called Ada. While Ada supports many of the features found in Fortran and Pascal, it has additional features making it easier and more reliable to use in large complex engineering software systems. Currently, the DOD is requiring that many new defense software systems be written in Ada.

To introduce you to Ada, we shall review the history of Ada's development and examine the primary constructs of the language. The philoso-

IEEE Center for The History of Electrical Engineering

A portrait of Augusta Ada Byron, Countess of Lovelace. Charles Babbages' proposed "Analytical Engine" would perform around 60 additions per minute when working. It never worked, but in 1843 Ada wrote "Observations on Mr. Babbage's Analytical Engine." There she gave examples of how the machine could be used including what has been considered the first "program," a sophisticated method of computing Bernoulli numbers.

0278-6648/85/1000-0026$01.00©1985 IEEE

Program Unit	Characteristic	Uses
Subprogram Procedure Function	Sequential action	Main program unit Perform operations and return none, one, or many values.
Package	Collection of resources	Group subprograms and declarations.
Task	Parallel processes	Performing operations which can occur concurrently.

TABLE I. ADA PROGRAM UNITS

phy behind Ada program design will be reviewed by designing and coding a simple program to add two matrices. Last, but not least, Ada's future prospects will be examined as related to both defense and nondefense applications.

Ada's history

In the early 1970s the United States Department of Defense noticed the trend of rising software costs and poor reliability for major defense systems. In 1975, the DOD established the Higher Order Language Working Group (HOLWG) to study the requirements for DOD programming languages, to evaluate existing languages against these requirements and to recommend a set of programming languages acceptable for use in DOD defense systems. Professionals from the DOD, industry and universities participated in this work. HOLWG's final proposal was to recommend the development of a new computer language which would be used for designing, implementing and maintaining all software aspects of defense systems. For lack of a better name, their language was called DOD-1.

To design such a new language was no small task. Recognizing this, the DOD chose to use an international design competition in which several teams would submit designs for DOD-1. The winning design was submitted by a French team from Honeywell/Honeywell Bull headed by Jean Ichbiah.

It was decided to change the name from DOD-1 to Ada, in honor of Augusta Ada Byron, Countess of Lovelace, and daughter of the poet Lord Byron. Ada Lovelace (1815–1851) was a mathematician who worked with Charles Babbage on his difference and analytic engines—two mechanical computers. She suggested programming Babbage's machines. For this work, she is considered the world's first programmer.

Following the design, the testing and evaluation period began. This process involved issuing a public invitation to take an existing application and to implement it in Ada. In addition, a preliminary language reference manual was developed and development of a compiler validation facility began.

As individual companies develop Ada compilers, they will be tested for their ability to meet the DOD standards for Ada. Compilers meeting these standards will be issued a validation number. DOD can then require that defense software systems be written to run using DOD validated Ada compilers. In this way, software systems developed on one type of hardware can be combined with software systems developed on another. Such software systems are referred to as transportable.

The story of Ada is far from complete. While a few companies such as Data General, Digital Equipment Corporation, and Telesoft have developed DOD validated Ada compilers, many others will soon be available. Work is also continuing on the development of software tools to be used in writing, debugging, and maintaining Ada programs. These tools constitute what is called the Ada Programming Support Environment (APSE). In time, these tools, like Ada, will be standardized so that the APSE available on all types of hardware will be the same.

One other very important issue is that of training Ada programmers. Since most engineers know Fortran and/or Pascal but not Ada, they will have to learn the new language. This represents a large investment for many companies. Young engineers who are seeking jobs and who know Ada will be particularly attractive to these companies.

An overview of Ada structure

An Ada program is composed of one or more program units, each may be compiled separately. These units are subprograms, packages, and tasks. Subprograms, procedural or functional, have characteristics and uses similar to their counterparts in Fortran and Pascal.

Packages provide a convenient way of grouping subprograms. Subprograms may be grouped according to their functions or because they share common data. For example, a group of subprograms used in circuit design and analysis might be combined into a single package. This concept is similar to that of program libraries in Fortran and Pascal.

Tasks are important in systems where processes occur in parallel. Consider a system which monitors a group of devices, performs analysis on the data from each, and prints out the results. Tasking would at the same time allow gathering data from one device, analyzing the data from a second device, and printing the results for a third device! While tasking is difficult with Fortran and Pascal it is easy to implement in Ada by simply developing a program unit for each of the three tasks. Ada does the rest.

Generally, each Ada program unit has two parts: the specification and the body. Unlike Fortran but like Pascal, all objects (variables and special constants) must be declared in Ada. This is done in the specification part of the unit. The body contains the Ada code used to perform the desired operations. Fig. 1 shows the basic parts of a procedure program unit. The name of the procedure is MATRIX_DEMO. Statements beginning with — are comments in Ada.

To declare an object in an Ada program, we must give both its name and its type. Examples of object types are integer, real (float or fixed), and array. The usual approach is to declare the types and then to declare the objects. We refer to objects as variables

```
procedure MATRIX_DEMO is
      --The specification part goes
      --in this section.
begin
      --The body of the procedure
      --goes in this section.
end MATRIX_DEMO
```

Fig. 1. Structure of an Ada procedure.

```
type COUNTER_TYPE is integer;
NUMBER_OF_DAYS: COUNTER_TYPE;

type PLANETARY_MEASUREMENT is digits 15;
ORBIT_LENGTH: PLANETARY_MEASUREMENT;

type MATRIX_TYPE is array
   (POSITIVE range <> ,
    POSITIVE range <> ) of INTEGER;
A,B: MATRIX_TYPE (1..2,1..3);

type MOTOR_STATE is (OFF, FORWARD, REVERSE);
HOIST: MOTOR_STATE:=OFF;
```

Fig. 2. Examples of Ada type and object declarations.

in Fortran and Pascal. Consider the examples shown in Fig. 2.

Line one defines a type called COUNTER_TYPE which is integer. Line two declares an object (variable) which is to hold a value of type COUNTER_TYPE (i.e. integer). Line three defines a real floating point type which has 15 digits of accuracy. The object ORBIT_LENGTH is then declared to be of type PLANETARY_MEASUREMENT. Line five declares MATRIX_TYPE to be a two-dimensional array of integers. Note that the subscripts are to be positive integers but line five gives no range of values for these subscripts. When the matrices A and B are declared in line six, the subscript ranges are specified. Hence, A and B are matrices of integer values each having two rows and three columns. MOTOR STATE is referred to as an enumeration type. Objects of type MOTOR_STATE can have one of three values; OFF, FORWARD or REVERSE. In line eight, HOIST is declared to be of type MOTOR_STATE and is initialized to OFF.

After declaring objects, we can then move to the body of the Ada program where we actually manipulate these objects.

Ada: a design tool and language

In addition to being a programming language, Ada also provides a tool for designing systems. This approach to design is called object-oriented. It is an improvement over existing program design methodologies in that it makes it easier to model real-world systems and provides a way to design largely self-documenting systems. That is, the design and development of a system are performed in such a way that the need for additional systems documentation is greatly reduced. Current research is being done on systems which can take object-oriented designs and turn them into Ada programs!

The basic components of the object-oriented design technique are:

1. Define the problem.
2. Develop an informal strategy for solving the problem.
3. Formalize the strategy.

The latter component consists of identifying objects of interest and their operations, establishing the interfaces among the various program units which process the objects and then implementing the operations.

To illustrate object-oriented design and to provide an introduction to the Ada programming language, the design and implementation of a simple function to add two matrices will be considered. The fundamentals of object-oriented design are followed.

1. Define the problem.
   Develop an Ada function which computes the sum of two matrices and returns the result.
2. Develop an informal strategy for solving the problem.

The sum of two matrices A and B is found by adding their corresponding elements. For example, if the result of $A + B$ is to be returned in a matrix called RESULT, we calculate $\text{RESULT}(I,J) = A(I,J) + B(I,J)$ for each element in A and B. All matrices must have the same number of rows and all matrices must have the same number of columns. We will assume that all matrix elements are integers and that this function will be called by a procedure.

3. Formalize the strategy.
   a. Identify the objects of interest and their operations.
   b. Establish the interfaces among the program units.
   c. Implement the operations.

We shall discuss each of these as it applies to our matrix addition problem.

The objects of interest are the three matrices; A, B and RESULT. These are matrices whose elements are integers. The operation is matrix addition.

Establishing the interfaces refers to creating the actual specification part of our function. Note that this is done in the initial stages of object-oriented design before consideration is given to the algorithms to be used in solving the problem. Fig. 3 shows how this interface might be declared.

Note that RESULT to be of MATRIX_TYPE is declared. It is to have the same range of rows and columns as matrix A. Matrices A and B along with MATRIX_TYPE will actually be declared in the procedure which calls MATRIX_SUM. Lines five and six in Fig. 2 show how this is done. MATRIX_SUM returns a value of MATRIX_TYPE.

The last step in designing this Ada function is to implement the actual

```
function MATRIX_SUM (A,B: MATRIX_TYPE)
 return MATRIX_TYPE is

--Declare RESULT

     RESULT: MATRIX_TYPE
       (A'RANGE(1),A'RANGE(2));

begin

     --The function body will
        be placed here later.

end MATRIX_SUM;
```

Fig. 3. MATRIX_SUM specification.

function; write the statements which make up the function body. Fig. 4 shows the completed function.

Note the form of the looping structure. A'RANGE(1) refers to the range of the first subscript for matrix A, i.e., the range of rows. A'RANGE(2) refers to the range of column subscripts for A. Ada subscripts may range over integers which are negative, zero and/or positive. Unlike Fortran, they may also range over enumeration types. The looping objects, I and J are not declared. Their

```
function MATRIX_SUM (A, B : MATRIX_TYPE)
 return MATRIX_TYPE is

     RESULT : MATRIX_TYPE
      (A'RANGE (1), A'RANGE (2));

begin
     for I in A'RANGE (1) loop
          for J in A'RANGE (2) loop
               RESULT (I, J)
                 := A (I, J) + B (I, J);
          end loop;
     end loop;
     return RESULT;
end MATRIX_SUM;
```

Fig. 4. MATRIX_SUM function.

values are undefined on exit from their respective loops. After exiting the outer loop, the value of result is returned as the value of the function.

The function can now be used by any Ada unit which needs to add matrices. Fig. 5 shows a simple procedure which initializes matrices *A* and *B*, performs *A* + *B*, and prints the result.

TEXT_IO referenced in lines one, two and nine is an Ada package of input/output procedures. Put and new_line are procedures in TEXT_IO.

In line eight, Ada has been told that the function MATRIX_SUM is in a separate file. Instead of putting MATRIX_SUM in a separate file, it could be included in the MATRIX_DEMO procedure. All that is needed is to replace line eight, with the statements shown in Fig. 4.

When the program is executed, statement 12 passes matrices *A* and *B* to the function MATRIX_SUM. Once the matrices are added, the results are returned and placed in matrix SUM. The program output is the matrix;

```
 6   8  10
12  14  16
```

Fig. 5 shows the same program written in UCSD Apple Pascal. In this figure, the matrix addition subprogram has been placed inside the main program. While UCSD Apple Pascal is somewhat more restrictive than other Pascal compilers, this program provides an interesting comparison betwen Pascal and Ada.

Ada's future

There are many features of Ada which have not been discussed. These features are designed to make Ada a powerful tool in designing, implementing and maintaining large software systems. The Department of Defense is not only requiring that many new defense systems be designed and programmed using Ada, but the National Aeronautics and Space Administration (NASA) is investigating the use of Ada for the many systems in the new space station to be launched in the early 1990's.

Although Ada is new and the number of DOD validated compilers are relatively few, results are beginning to show that Ada does indeed simplify the design, implementation and maintenance of large, complex software systems. The transportability of code coupled with the object-oriented design approach has been responsible for improving communication and understanding between all of the people involved in the design and implementation of large systems.

The use of Ada in nondefense applications is expected to be somewhat slower than it has been in defense applications. One reason for this is the current short supply of Ada programmers. Another reason is the unavailability of auxiliary software which interfaces to Ada. For example, a program requiring special data handling capabilities will need to be interfaced to a database management system. As Ada matures, these problems will disappear and Ada will enjoy wide use in nondefense areas.

While Ada is not the programming language to end all programming languages, it is one of the best tools we have for dealing with large, complex software systems. In this respect, we can expect it to be around for some time to come.

Read more about it

- Booch and Grady, *Software Engineering with Ada*, Benjamin/Cummings, 1983.
- Barnes, J. G. P., *Programming in Ada*, 2nd ed., Addison-Wesley, 1982.
- *Reference Manual for the Ada Programming Languages*, ANSI-MIL-STD-1815A-1983, American National Standards Institute.

RR Software and Telesoft sell Ada compilers which run on some of the more popular microcomputers. These compilers are not DOD validated nor do they provide all the features of Ada.

```
with TEXT_IO;
use TEXT_IO;
procedure MATRIX_DEMO is

  type MATRIX_TYPE is array
   (POSITIVE range <>, POSITIVE range <>)
     of INTEGER;

  A   : MATRIX_TYPE (1 .. 2, 1 .. 3)
        := ((1, 2, 3), (4, 5, 6));
  B   : MATRIX_TYPE (1 .. 2, 1 .. 3)
        := ((5, 6, 7), (8, 9, 10));
  SUM : MATRIX_TYPE (1 .. 2, 1 .. 3);

  function MATRIX_SUM (A, B : MATRIX_TYPE)
    return MATRIX_TYPE is separate;

  package INT_IO is new
    TEXT_IO.INTEGER_IO (INTEGER);
  use INT_IO;

begin

  SUM := MATRIX_SUM (A, B);

  for I in SUM'RANGE (1) loop
    for J in SUM'RANGE (2) loop
         put (SUM (I, J));
    end loop;
    new_line;
  end loop;
  new_line (2);

end MATRIX_DEMO;
```

Fig. 5. Ada procedure which uses MATRIX_SUM.

About the author

Daniel C. St. Clair is Professor of Computer Science at the University of Missouri-Rolla Graduate Engineering Center in St. Louis, Missouri. Dr. St. Clair has spent two summers with NASA's Johnson Space Center in Houston, Texas where he worked on space station projects involving Ada, graphics and expert systems. He has been actively involved with McDonnell-Douglas Corporation in evaluating potential commercial applications of Ada. □