The Language Ada and Concurrent Processes

Jeffrey Kurt Lovelace

"The Language Ada and Concurrent Processes"

written and designed by

Jeffrey K. Lovelace

under the direction of

Dr. Albert Crawford

Department of Computer Science

Southern Illinois University at Carbondale

for

UHON 499

Senior Project

In the following text, I would like to reflect upon my learning experience of "The Language Ada and Concurrent Processes."

When selecting a project, I chose an area dealing with three very important and widely used items in the field of computer science. These are, the UNIX operating system, the language Ada, and concurrency programming. I hope to apply this knowledge directly to the field which I venture into.

I felt this independent undertaking had to be a project which would reflect the culmination of my gained knowledge since starting college, and at the same time, show my aspiration to set high goals and achieve them. In some aspects, I have gone beyond the goals I have set, and in others, I have fallen short.

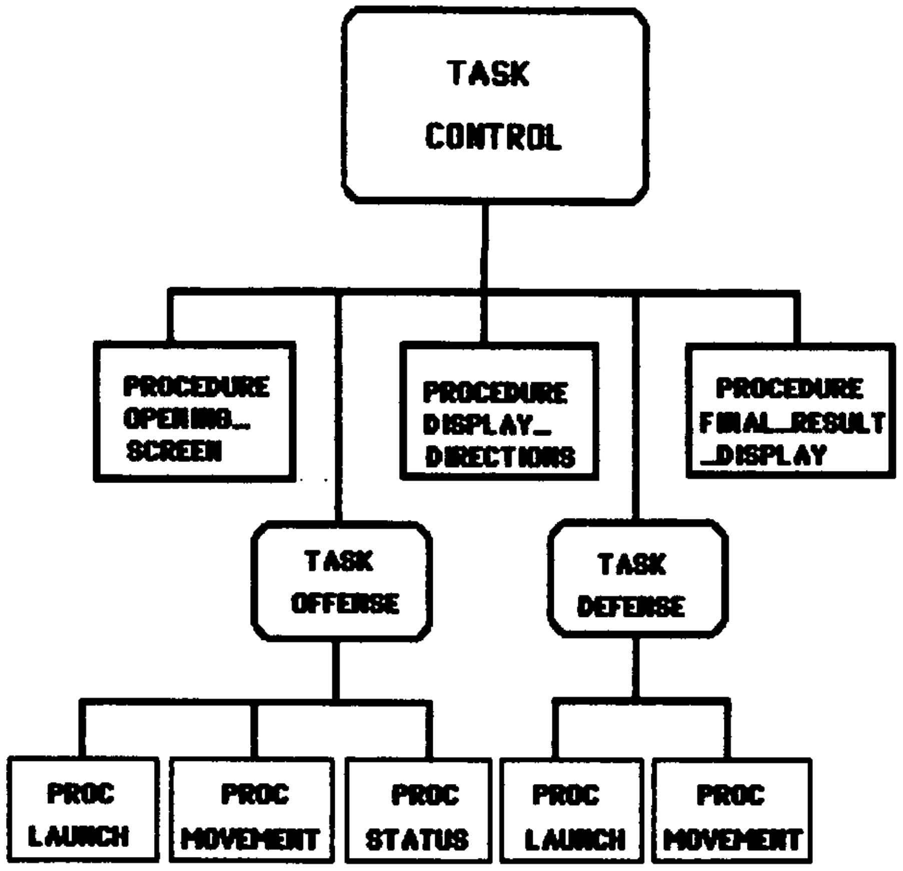
I began by finding an "expert" to lead me in my endeavor. At that time, very few doctors in the department were educated in the language Ada. At times, I found that to be a hinderance, but nonetheless, Dr. Albert Crawford inspired me to continue on my independent journey. After selecting a specific topic, I spent the summer of 1988 reading various texts on Ada, so I would have a competent working knowldege of the language Ada before beginning my programming assignment in the fall of 1988. With some experience gained in Ada from a previous course and the time devoted over the summer, I felt confident enough to begin my project.

Dr. Crawford and I sat down and devised an agenda to follow for the fall semester. The project would be broken down into two phases. The first phase would introduce me to the UNIX system, the language Ada, and concurrent programming, and the second phase would test my ability to design a mult-tasking program.

To briefly describe the first phase, the two task simulation involved a dog chasing a cat in a one hundred yard by one hundred yard field. The user would enter the two animals starting locations (i.e. the X and Y coordinates of each), and their speeds. (Note: The dog is always faster than the cat) Once these pieces of information were inserted into the system, the simulation would begin. During the process of the simulation, there is a constant visual display of the X,Y coordinates for each animal. The simulation ends when the dog is less than six feet from the cat. At the termination of both task, a report of the final coordinates and the time it took for capture is displayed.

Now to describe the more complicated second phase. This phase dealt with a multi-tasking strategic defense simulation. The premise behind this phase is that a country is considering strengthening their defensive capabilities. They would like to purchase a space based defensive missle platform which would deter or destroy any attack on their country. The primary contractor boasts a high success rate, but the country is unsure of the true peformance (refer to Table 2 - Performance Specifications p. 11). To provide more reliable information before buying the multi-billion dollar weapon, they requested the design of a computer software package which would simulate the defensive weapon. By using this simulation, they could insert different "scenarios" into the system to see if the weapon performed to specifications. By using this software package, they could make a better educated decision before investing the billions of dollars.

To describe the package design in more detail, it is broken down into four blocked units, a missle task, a bomber task, a process control task, and a screen I/O task. To begin a description of each task and its function, I will start at the highest level in the hierarchy and work down.



Task Control is a designed block to control the flow of the multi-tasking program, obtain variant values from the user, display directions upon request, start up lower level task, and return final result values to the user. This unit is the central manager of the software package.

Task Offense and Defense are essentially of the same purpose except for opposite sides (refer to Table 1 - Specifications p.10). The offensive task attempts to guide the nuclear equipped bomber through the one hundred mile range of the defensive space platform without being hit. (Note: the offensive weapon in this simulation has been referred to as a nuclear equipped bomber, but it could be any other type of offensive weapon) On the other side, the defensive task attempts to track and destroy its target. Each task is considered an individual unit (i.e. one plane or one missle), and the user defines how many copies of each task are to be executed. These two tasks invoke launch and movement procedures, and the offensive task invokes an extra procedure referred to as status. Now that the system has been described in an outlined form, I would like to move on to the actual progress report of the learning experience.

During the process of designing and implementing the two phases, I quickly grasped and completed phase one, but struggled in phase two. Phase one came together quickly as I became acquainted with the concepts and syntax of writing concurrent programs. No major stumbling blocks occurred, and the project was finished on time. I cannot say the same for the second phase.

Phase two started off fairly quick with the initial defining of packages and task. I then began coding, and found that I needed to devote more time in studying the design and implementation of tasks. After checking out a few reference books (refer to reference table p. 8-9), I became more adept at writing code for tasks. One weakness I felt I had was moving from sequential design thought processes to concurrent design thought processes. The references I had read, described tasks and how they interact with other tasks, but did not define how to design a multi-task environment. I continued the hand coding process for about four weeks.

After successfully hand tracing the program several times, I went to the lab to begin entry and testing of each module. Building test harnesses for each block was time consuming and slow, but I felt this was the best direction to proceed in. As progress continued, I began to realize some additions and modifications I was going to need in order for the system to handle all possibilities. This again detracted me from the original time schedule.

Another problem I found to be a hinderance to my progress was the lack of experienced people in the language Ada to reference. As I mentioned earlier, Dr. Crawford and one graduate student were the only people in the department having experience in the language Ada. At times, I found myself spending hours in the Ada reference manual trying to track down syntax errors.

Upon reaching the deadline for the project, I had about three-fourths of the basic software package up and running, The software package at the time of the deadline, would allow the user to input the variables into the system, and in about 75% of the possible conditions tested, the defensive units would track their targets successfully. The other 25% had problems in following their targets to completion. I suspect those 25% were either getting lost or the target was returning invalid coordinates.

In closing, this has been a very valuable learing experience in several perspectives. I have learned to be better prepared when designing multi-task environments, more so than in sequential programming. I have learrned how to deal with the difficulty of having to rely primarily on text references rather than people. Finally, I have experienced a first hand dealing with UNIX, Ada, and concurrency programming. With this experience, I feel I have gained an additional educational experience in which the current undergraduate curriculum does not offer.

REFERENCE TABLE & DESCRIPTION

"Discrete - Event Simulation"
by Jerry Banks & John S. Carlson, II

Main reference for use in simulations and how they work, when to use simulation, and why they are useful. The text described when simulation is an appropriate tool and gave several examples.

"Ada* as a Second Language"
by Norman H. Cohen

Main reference used in studying the language Ada. Several items were convered in this text. I learned about the following items:

Basic constructs of the language Ada
Compilation units
Subprogram
Statements
Standard and private types
Tasks
Packages

Once I had completed reading this text, I concentrated on learning the concepts of concurrency programming. I used this text as my initial reference for learning about task, task bodies, task types, rendezvous, entry calls, accepts statements, and activation and termination of task. I felt after completing this text, I still needed futher references and examples of how to write tasks.

"Concurrent programming in Ada*"
by Alan Burns

This text was read after convering "Ada* as a Second Language." This reference gave me a more intense view of tasking, specifically dealing with inter-process communication
i.e. Synchronization
Deadlock & Indefinite Postponements

System Performance and Reliability
Shared Variables
Rendezvous

and inter-task communication
i.e. Entry statement
Accept statement
Select statement.

"The UNIX Programming Environment"**
by Brian W. Kernighan and Rob Pike

Use this reference to learn the basic features and fundamental properties about the UNIX programming environment.

"Ada* primer"
by Philip I. Johnson

I used this text primarily during the first phase, "Cat & Dog." This text helped me with syntax and general development of the first project.

"Good Programming Practice in Ada"
by P.A. Luker

This reference was used during the second phase, "Strategic Defense Simulation." This text provided additional examples of packages, tasks, type declarations, blocking, and I/O. I found this book to be written with a strong reference to the language Pascal which proved to be an advantage for me. I have mastered the language Pascal thoroughly and programmed in it for the last five years. This text helped me to understand some of the more complicated features of Ada by referencing the language Pascal.

* Ada is a registered trademark of the U.S. Government (Ada Joint Programming Office)

** UNIX is a trademark of Bell Laboratories

TABLE 1 - SPECIFICATIONS

	OFFENSIVE	DEFENSIVE	COMMENTS
SPEED	1400 mph = 2053.33 ft/sec	1500 mph = 2199.99 ft/sec	Defensive weapon reaches MACH 2
RANGE	1000+ miles	100 miles	Defense launch target < 100 m
POSITION	100 miles above earths surface	5 miles above earths surface	These are intial positions
MODE	EVASIVE	TRACKING	

TABLE 2 : PERFORMANCE SPECIFICATIONS

DEFENSIVE MISSLE PLATFORM

MAX # OF DEFENSIVE MISSLES	100	FUTURE EXPANSION PLANNED
MAX RANGE PER MISSLE	100 MILES	FUTURE EXPANSION CAPABILITIES
MAX TRACKING ABILITY AT SPECIFIC INSTANCE	< 30 TARGETS	THIS IS A RESTRICTION OF THE HARDWARE. FUTURE EXPANSION POSSIBLE.
RELOADING CAPABILITIES	NONE	PLATFORM IS REUSABLE. WOULD NEED TO BE REFILLED BY AUXILARY UNIT.
MISSLE LAUNCH DELAY BETWEEN CYCLES	.025 SECONDS	MAX DELAY IN WORST CASE: 100 MISSLES = 2.5 SEC OFFENSIVE DISTANCE = 0.97 MILES

```
--###############################################################
--#                                                             #
--#  NAME          :  JEFFREY K. LOVELACE                       #
--#  COURSE        :  UHON 499                                  #
--#  TITLE         :  "THE LANGUAGE ADA AND CONCURRENT PROCESSES" #
--#  PHASE         :  1 - DOG CHASE CAT SIMULATION              #
--#                                                             #
--#  CHAIRPERSON   :  DR. ALBERT CRAWFORD                       #
--#  DEPARTMENT    :  COMPUTER SCIENCE                          #
--#                                                             #
--###############################################################
with TEXT_IO;
use  TEXT_IO;

procedure DOG_CHASE_CAT is

   package REAL_IO is new FLOAT_IO(FLOAT);
   use     REAL_IO;

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

   CAT_X     :  INTEGER;
   CAT_Y     :  INTEGER;
   DOG_X     :  INTEGER;
   DOG_Y     :  INTEGER;
   CAT_SPEED :  INTEGER;
   DOG_SPEED :  INTEGER;
   TIME      :  FLOAT;
   INCREMENT :  FLOAT := 0.25;
   CH        :  CHARACTER;

--###############################################################
--#                                                             #
--#                     T A S K   S E T U P                     #
--#                                                             #
--#  This task is used to retrieve the necessary information from #
--#  the user to begin the simulation.                          #
--#                                                             #
--#  START                       releases task to begin execution #
--#  INITIAL_MEOW                passes initial location of cat #
--#                              to calling routine             #
--#  INITIAL_BARK                passes initial location of dog #
--#                              to calling routine             #
--#                                                             #
--###############################################################
   task SETUP is
        entry START;
        entry INITIAL_MEOW(CAT_X,CAT_Y : INTEGER);
        entry INITIAL_BARK(DOG_X,DOG_Y : INTEGER);
   end SETUP;


--###############################################################
--#                                                             #
--#                       T A S K   C A T                       #
--#                                                             #
--#  This task represents the functioning of the cat during the #
--#  simulation.                                                #
--#                                                             #
--#  START                       releases task to begin execution #
--#  LAST_MEOW                   passes cat's current location to #
```

```
--*                              calling routine                  *
--*                                                               *
--*****************************************************************
   task CAT is
      entry START;
      entry LAST_MEOW(CAT_X,CAT_Y : INTEGER);
   end CAT;


--*****************************************************************
--*                                                               *
--*                        T A S K   D O G                        *
--*                                                               *
--*  This task represents the functioning of the dog during the  *
--*  simulation.                                                  *
--*                                                               *
--*  START                       releases task to begin execution *
--*  LAST_BARK                   passes dog's current location to *
--*                              calling routine                  *
--*                                                               *
--*****************************************************************
   task DOG is
      entry START;
      entry LAST_BARK(DOG_X,DOG_Y : INTEGER);
   end DOG;


--*****************************************************************
--*                                                               *
--*                       T A S K   F I N A L                     *
--*                                                               *
--*  This task prints out the location of where the dog captured  *
--*  the cat, and how much time it took.                          *
--*                                                               *
--*  FINISHED                    releases the task when task DOG  *
--*                              and task CAT have terminated     *
--*                                                               *
--*****************************************************************
   task FINAL is
      entry FINISHED;
   end FINAL;


--*****************************************************************
--*                                                               *
--*            P R O C E D U R E   C L E A R _ S C R E E N        *
--*                                                               *
--*****************************************************************
   procedure CLEAR_SCREEN is

   begin

      for I in 1..25 loop
         NEW_LINE;
      end loop;

   end CLEAR_SCREEN;


--*****************************************************************
--*                                                               *
--*                  T A S K   B O D Y   F I N A L                *
--*                                                               *
--*****************************************************************
   task body FINAL is
   begin

      accept FINISHED;
      accept FINISHED;

      CLEAR_SCREEN;

      PUT_LINE("!#*%@, CRUNCH, CRUNCH, the dog caught the cat");
      PUT("at :");
      PUT(DOG_X,3);
      PUT(",");
      PUT(DOG_Y,3);
      NEW_LINE;
      NEW_LINE;
      PUT("It took the dog");
      PUT(TIME,4,2,0);
      PUT_LINE(" seconds to catch the cat");

   end FINAL;


--*****************************************************************
--*                                                               *
--*                T A S K   B O D Y   S E T U P                  *
--*                                                               *
--*****************************************************************
   task body SETUP is

   begin

   accept START;

   PUT_LINE("Welcome to a computer simulation of a dog chasing a");
   PUT_LINE("cat in a field.  In this simulation, the computer will");
   PUT_LINE("execute the dog catching the cat much faster than in");
   PUT_LINE("real time, but at the end of the chase, the actual");
   PUT_LINE("location and amount of real time elapsed will be");
   PUT_LINE("displayed.  The field, in real space is 100 yards by");
   PUT_LINE("100 yards, which will be represented by a 100 x 100");
   PUT_LINE("matrix grid.  As in real life, if the cat hits a boundary,");
   PUT_LINE("or the fence, the cat will have to move along the fence,");
   PUT_LINE("and cannot escape.  During the chase, the distance between");
   PUT_LINE("the dog and the cat will be displayed.  So lets get started!");
   NEW_LINE;
   PUT_LINE("HIT ANY KEY AND <return> TO CONTINUE...");

   GET(CH);

   CLEAR_SCREEN;

   PUT_LINE("ENTER THE CAT'S X POSITION (1-100)");
   GET(CAT_X);
   NEW_LINE;
   PUT_LINE("ENTER THE CAT'S Y POSITION (1-100)");
   GET(CAT_Y);

   CLEAR_SCREEN;

   PUT_LINE("Now before entering the dog's position, we ask that you");
   PUT_LINE("place the dog at least 25 yards away from the cat in");
   PUT_LINE("order to give it a sporting chance--but nonetheless, it");
   PUT_LINE("is not necessay");
   PUT_LINE("HIT ANY KEY AND <return> TO CONTINUE....");

   GET(CH);
```

```
      CLEAR_SCREEN;

      PUT("REMEMBER, THE CAT IS AT ");
      PUT(CAT_X);
      PUT(",");
      PUT(CAT_Y);
      NEW_LINE;
      PUT_LINE("ENTER THE DOG'S X POSITION (1-100)");
      GET(DOG_X);
      NEW_LINE;
      PUT_LINE("ENTER THE DOG'S Y POSITION (1-100)");
      GET(DOG_Y);

      CLEAR_SCREEN;

      PUT_LINE("Now we realize there are various kinds of cats and dogs,");
      PUT_LINE("so we are going to allow you to enter their types. ");
      PUT_LINE("Remember, the dog will always be faster than the cat.");
      PUT_LINE("HIT ANY KEY AND <return> TO CONTINUE....");

      GET(CH);

      CLEAR_SCREEN;

      PUT_LINE("ENTER THE CATS SPEED (3-5)");
      PUT_LINE("3 = CAT WITH 3 LEGS");
      PUT_LINE("4 = STANDARD CAT   ");
      PUT_LINE("5 = CAT IN PURSUIT OF DINNER");
      GET(CAT_SPEED);

      CLEAR_SCREEN;

      PUT_LINE("ENTER THE DOGS SPEED (6-8)");
      PUT_LINE("6 = POODLE");
      PUT_LINE("7 = GOLDEN RETRIEVER");
      PUT_LINE("8 = SALUKI HUNTING DOG");
      GET(DOG_SPEED);

      CLEAR_SCREEN;

      PUT("OK, the cat is at ");
      PUT(CAT_X);
      PUT(",");
      PUT(CAT_Y);
      NEW_LINE;
      PUT("and the dog is at ");
      PUT(DOG_X);
      PUT(",");
      PUT(DOG_Y);
      NEW_LINE;
      PUT("and we have a");

      if DOG_SPEED = 6  then
         PUT(" poodle");
      elsif DOG_SPEED = 7 then
         PUT(" golden retriever");
      else
         PUT(" saluki hunting dog");
      end if;

      PUT(" chasing a");

      if CAT_SPEED = 3 then
         PUT_LINE(" three-legged cat.");
      elsif CAT_SPEED = 4 then
         PUT_LINE(" standard cat.");
```

```
   else
      PUT_LINE(" cat in pursuit of dinner.");
   end if;

   NEW_LINE;
   PUT_LINE("HERE WE GO........");
   NEW_LINE;
   PUT_LINE("HIT ANY KEY AND <return> TO CONTINUE...");
   GET(CH);

   CAT.START;
   DOG.START;

   accept INITIAL_MEOW(CAT_X,CAT_Y : INTEGER);

   accept INITIAL_BARK(DOG_X,DOG_Y : INTEGER);

   accept INITIAL_BARK(DOG_X,DOG_Y : INTEGER);

   end SETUP;


--*******************************************************************
--*                                                                 *
--*                   T A S K   B O D Y   C A T                     *
--*                                                                 *
--*******************************************************************
   task body CAT is

      NUMER             : INTEGER;
      DENOM             : INTEGER;
      SLOPE             : FLOAT;
      NUMER_MULTIPLIER : FLOAT;
      DENOM_MULTIPLIER : FLOAT;
      X1,X2,Y1,Y2       : INTEGER;

   begin

      accept START;

      SETUP.INITIAL_MEOW(CAT_X,CAT_Y);

      SETUP.INITIAL_BARK(DOG_X,DOG_Y);

CATLOOP:

   loop

   exit when (abs(CAT_X - DOG_X) < 2) and (abs(CAT_Y - DOG_Y) < 2);

      PUT("The cat is at ");
      PUT(CAT_X);
      PUT(" , ");
      PUT(CAT_Y);
      NEW_LINE;

      PUT("The dog is at ");
      PUT(DOG_X);
      PUT(" , ");
      PUT(DOG_Y);

      NEW_LINE;
      NEW_LINE;

      delay 2.0;
```

```
if (DOG_Y = CAT_Y) and (DOG_X < CAT_X) then

if (CAT_X + CAT_SPEED <= 100) then
   CAT_X := CAT_X + CAT_SPEED;
elsif (CAT_Y + CAT_SPEED <= 100) then
   CAT_Y := CAT_Y + CAT_SPEED;
else
   CAT_Y := CAT_Y - CAT_SPEED;
end if;

elsif (DOG_Y = CAT_Y) and (DOG_X >= CAT_X) then
   if (CAT_X - CAT_SPEED >= 1) then
      CAT_X := CAT_X - CAT_SPEED;
   elsif (CAT_Y - CAT_SPEED >= 1) then
      CAT_Y := CAT_Y - CAT_SPEED;
   else
      CAT_Y := CAT_Y + CAT_SPEED;
   end if;
end if;

if (DOG_X = CAT_X) and (DOG_Y < CAT_Y) then
   if (CAT_Y + CAT_SPEED <= 100) then
      CAT_Y := CAT_Y + CAT_SPEED;
   elsif (CAT_X + CAT_SPEED <= 100) then
      CAT_X := CAT_X + CAT_SPEED;
   else
      CAT_X := CAT_X - CAT_SPEED;
   end if;
elsif (DOG_X = CAT_X) and (DOG_Y >= CAT_Y) then
   if (CAT_Y - CAT_SPEED >= 1) then
      CAT_Y := CAT_Y - CAT_SPEED;
   elsif (CAT_X + CAT_SPEED <= 100) then
      CAT_X := CAT_X + CAT_SPEED;
   else
      CAT_X := CAT_X - CAT_SPEED;
   end if;
end if;

if (DOG_X /= CAT_X) and (DOG_Y /= CAT_Y) then
   X1 := CAT_X;
   X2 := DOG_X;
   Y1 := CAT_Y;
   Y2 := DOG_Y;

   NUMER := Y2 - Y1;
   DENOM := X2 - X1;

   SLOPE := float(NUMER) / float(DENOM);

   if (SLOPE < 1.0) and (SLOPE > 0.0) then
      NUMER_MULTIPLIER := SLOPE;
      DENOM_MULTIPLIER := 1.0 - SLOPE;
   elsif (abs(NUMER) = abs(DENOM)) then
         NUMER_MULTIPLIER := 0.5;
         DENOM_MULTIPLIER := 0.5;
   else
      DENOM_MULTIPLIER := abs(float(DENOM) / float(NUMER));
      NUMER_MULTIPLIER := 1.0 - DENOM_MULTIPLIER;
   end if;

   if (SLOPE > 0.0) and (NUMER > 0) then
      if (CAT_X - integer(float(CAT_SPEED) * DENOM_MULTIPLIER) >= 1) and
         (CAT_Y - integer(float(CAT_SPEED) * NUMER_MULTIPLIER) >= 1) then

         CAT_X := CAT_X - integer(float(CAT_SPEED) * DENOM_MULTIPLIER);
         CAT_Y := CAT_Y - integer(float(CAT_SPEED) * NUMER_MULTIPLIER);
```

```
          else
             CAT_X := CAT_X + CAT_SPEED;
          end if;

       elsif (SLOPE > 0.0) and (NUMER < 0) THEN
          if (CAT_X + integer(float(CAT_SPEED) * DENOM_MULTIPLIER) <= 100) and
             (CAT_Y + integer(float(CAT_SPEED) * NUMER_MULTIPLIER) <= 100) then

             CAT_X := CAT_X + integer(float(CAT_SPEED) * DENOM_MULTIPLIER);
             CAT_Y := CAT_Y + integer(float(CAT_SPEED) * NUMER_MULTIPLIER);

          else
             CAT_X := CAT_X - CAT_SPEED;
          end if;

       elsif (SLOPE < 0.0) and (NUMER < 0) then
          if (CAT_X - integer(float(CAT_SPEED) * DENOM_MULTIPLIER) >= 1) and
             (CAT_Y + integer(float(CAT_SPEED) * NUMER_MULTIPLIER) <= 100) then

             CAT_X := CAT_X - integer(float(CAT_SPEED) * DENOM_MULTIPLIER);
             CAT_Y := CAT_Y + integer(float(CAT_SPEED) * NUMER_MULTIPLIER);

          else
             CAT_Y := CAT_Y - CAT_SPEED;
          end if;

       elsif (SLOPE < 0.0) and (DENOM < 0) then
          if (CAT_X + integer(float(CAT_SPEED) * DENOM_MULTIPLIER) <= 100) and
             (CAT_Y - integer(float(CAT_SPEED) * NUMER_MULTIPLIER) >= 1) then

             CAT_X := CAT_X + integer(float(CAT_SPEED) * DENOM_MULTIPLIER);
             CAT_Y := CAT_Y - integer(float(CAT_SPEED) * NUMER_MULTIPLIER);

          else
             CAT_Y := CAT_Y + CAT_SPEED;
          end if;

       end if;

    end if;

    accept LAST_MEOW(CAT_X,CAT_Y : INTEGER);

    DOG.LAST_BARK(DOG_X,DOG_Y);

  end loop CATLOOP;

  FINAL.FINISHED;

  end CAT;


--*********************************************************************
--*                                                                   *
--*                 T A S K   B O D Y   D O G                         *
--*                                                                   *
--*********************************************************************
  task body DOG is

  X1,Y1,X2,Y2       : INTEGER;
  SLOPE             : FLOAT;
  NUMER             : INTEGER;
  DENOM             : INTEGER;
  NUMER_MULTIPLIER : FLOAT;
  DENOM_MULTIPLIER : FLOAT;
```

```
begin

   accept START;

   SETUP.INITIAL_BARK(DOG_X,DOG_Y);

   TIME := 0.0;

DOGLOOP:

   loop

   exit when (abs(CAT_X - DOG_X) < 2) and (abs(CAT_Y - DOG_Y) < 2);

   CAT.LAST_MEOW(CAT_X,CAT_Y);

   X1 := CAT_X;
   X2 := DOG_X;
   Y1 := CAT_Y;
   Y2 := DOG_Y;

   NUMER := Y2 - Y1;
   DENOM := X2 - X1;

   SLOPE := float(NUMER) / float(DENOM);

   if (SLOPE < 1.0) and (SLOPE > 0.0) then


      NUMER_MULTIPLIER := SLOPE;
      DENOM_MULTIPLIER := 1.0 - SLOPE;

   elsif (abs(NUMER) = abs(DENOM)) then

      NUMER_MULTIPLIER := 0.5;
      DENOM_MULTIPLIER := 0.5;

   else

      DENOM_MULTIPLIER := abs(float(DENOM) / float(NUMER));
      NUMER_MULTIPLIER := 1.0 - DENOM_MULTIPLIER;

   end if;

   if (SLOPE > 0.0) and (NUMER > 0) then

      DOG_X := DOG_X - integer(float(DOG_SPEED) * DENOM_MULTIPLIER);
      DOG_Y := DOG_Y - integer(float(DOG_SPEED) * NUMER_MULTIPLIER);

   elsif (SLOPE > 0.0) and (NUMER < 0) then

      DOG_X := DOG_X + integer(float(DOG_SPEED) * DENOM_MULTIPLIER);
      DOG_Y := DOG_Y + integer(float(DOG_SPEED) * NUMER_MULTIPLIER);

   elsif (SLOPE < 0.0) and (NUMER < 0) then

      DOG_X := DOG_X - integer(float(DOG_SPEED) * DENOM_MULTIPLIER);
      DOG_Y := DOG_Y + integer(float(DOG_SPEED) * NUMER_MULTIPLIER);

   elsif (SLOPE < 0.0) and (DENOM < 0) then

      DOG_X := DOG_X + integer(float(DOG_SPEED) * DENOM_MULTIPLIER);
      DOG_Y := DOG_Y - integer(float(DOG_SPEED) * NUMER_MULTIPLIER);

   end if;
```

```
      TIME := TIME + INCREMENT * float(DOG_SPEED);

      accept LAST_BARK(DOG_X,DOG_Y : INTEGER);

   end loop DOGLOOP;

   FINAL.FINISHED;

   end DOG;


--**********************************************************************
--*                                                                    *
--*            M A I N      P R O G R A M                              *
--*                                                                    *
--**********************************************************************
begin

PUT_LINE("Welcome to a simulation of a dog chasing a cat!");
NEW_LINE;
PUT_LINE("HIT ANY KEY AND <return> TO CONTINUE...");
GET(CH);
CLEAR_SCREEN;

SETUP.START;

end DOG_CHASE_CAT;
```

```
--*****************************************************************
--*                                                               *
--*  NAME         :  JEFFREY K. LOVELACE                          *
--*  COURSE       :  UHON 499                                     *
--*  TITLE        :  "THE LANGUAGE ADA AND CONCURRENT PROCESSES"  *
--*  PHASE        :  2 - A STRATEGIC DEFENSE SIMULATION           *
--*                                                               *
--*  CHAIRPERSON  :  DR. ALBERT CRAWFORD                          *
--*  DEPARTMENT   :  COMPUTER SCIENCE                             *
--*                                                               *
--*****************************************************************
with TEXT_IO; use TEXT_IO;
with CALENDAR; use CALENDAR;
with IO_EXCEPTIONS; use IO_EXCEPTIONS;

procedure MAIN is

   subtype STRING30 is STRING(1..30);

   package REAL_IO is new FLOAT_IO (FLOAT);
   use REAL_IO;

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

   package MISSLE is
      procedure LAUNCH( MIS_X, MIS_Y, MIS_Z : out FLOAT; NUMTASK : in INTEGER );
      procedure MOVEMENT( DISTANCE, MIS_X, MIS_Y, MIS_Z : in out FLOAT; NUMTASK : in INTEGER );
      procedure STATUS( ELAPSED_TIME : in DURATION; HIT_STATUS : out BOOLEAN );
   end MISSLE;

   package BOMBER is
      procedure LAUNCH( BOM_X, BOM_Y, BOM_Z : out FLOAT );
      procedure MOVEMENT( BOM_X, BOM_Y, BOM_Z : in out FLOAT; NUMTASK : in INTEGER );
   end BOMBER;

   package PROCESS_CONTROL is
      procedure OPENING_SCREEN;
      procedure DISPLAY_DIRECTIONS;
      procedure FINAL_RESULT_DISPLAY;
   end PROCESS_CONTROL;


--*****************************************************************
--*                                                               *
--*          T A S K  T Y P E   D E F E N S E _ T Y P E           *
--*                                                               *
--*  This task type is used for the space based missle group.     *
--*  During execution, there are two entry points into a defensive *
--*  task.  They are:                                             *
--*                                                               *
--*   START                  releases a task to begin execution   *
--*   MISSLE_POSITION        passes missle coordinates to calling  *
--*                          routine                              *
--*                                                               *
--*****************************************************************
   task type DEFENSE_TYPE is
      entry START( NUMTASK : in INTEGER );
      entry MISSLE_POSITION( DISTANCE, MIS_X, MIS_Y, MIS_Z : out FLOAT );
   end DEFENSE_TYPE;
```

```
--*************************************************************
--*                                                           *
--*          T A S K  T Y P E   O F F E N S E _ T Y P E       *
--*                                                           *
--*  This task type is used for the incoming nuclear bombers.  *
--*  During exectuion, there are two entry points into an      *
--*  offensive task.  They are:                                *
--*                                                           *
--*   START               releases a task to begin execution   *
--*   BOMBER_POSITION     passes bomber coordinates to calling *
--*                       routine                              *
--*                                                           *
--*************************************************************
   task type OFFENSE_TYPE is
      entry START( NUMTASK : in INTEGER );
      entry BOMBER_POSITION( BOM_X, BOM_Y, BOM_Z : out FLOAT );
   end OFFENSE_TYPE;


--*************************************************************
--*                                                           *
--*                    T A S K   C O N T R O L                *
--*                                                           *
--* This task acts as the central manager for the execution of the *
--* simulation.  It controls retrieval of information from the *
--* user, initiates a direction display (if requested), start up *
--* lower level task (Offense & Defense), and calls for the final *
--* display of information resulting from the simulation.     *
--*                                                           *
--*************************************************************
   task CONTROL;


--*************************************************************
--*                                                           *
--*                  T A S K   S C R E E N _ I O              *
--*                                                           *
--*  This task controls all printing to the screen once tasks *
--*  begin execution.  The calling task makes a request to send *
--*  output to the screen via the SCREEN_IO.LOCK.  If the task *
--*  is not currently in use, the SCREEN_IO task will honor the *
--*  call and print the information sent from the calling task. *
--*  Once the calling task is finished, it will UNLOCK the     *
--*  SCREEN_IO task, in which case, the SCREEN_IO task is free *
--*  to honor other calling tasks requests.                    *
--*                                                           *
--*************************************************************
   task SCREEN_IO is
      entry LOCK;
      entry STRING_IO( OUT_LINE : STRING30 );
      entry INT_IO( OUT_INT : INTEGER );
      entry REAL_IO( OUT_REAL : FLOAT );
      entry UNLOCK;
   end SCREEN_IO;


   BOM_MAX_SPEED : constant FLOAT := 2053.333;
   MIS_MAX_SPEED : constant FLOAT := 2199.999;
   OFFENSE       : array(1..10) of OFFENSE_TYPE;
   DEFENSE       : array(1..10) of DEFENSE_TYPE;
   NUMBOM        : INTEGER;
   NUMMIS        : INTEGER;

--*************************************************************
--*                                                           *
--*                    F U N C T I O N   S Q R T                *
--*                                                              *
--*  This is a function created to handle the task of taking the *
--*  square root of a real number.                               *
--*                                                              *
--****************************************************************
   function SQRT( X : in FLOAT; EPS : in FLOAT := 0.01 ) return FLOAT is

   OLD_VALUE : FLOAT;
   NEW_VALUE : FLOAT;

   begin

      OLD_VALUE := 0.0;
      NEW_VALUE := X/2.0;

      while abs(NEW_VALUE - OLD_VALUE) > EPS loop

         OLD_VALUE := NEW_VALUE;
         NEW_VALUE := 0.5 * (OLD_VALUE + X/OLD_VALUE);

      end loop;

      return NEW_VALUE;

   end SQRT;


--****************************************************************
--*                                                              *
--*         F U N C T I O N   R A N D O M _ V A L U E            *
--*                                                              *
--****************************************************************
   function RANDOM_VALUE (DIRECTION_CHAR : in CHARACTER) return FLOAT is

      NEW_VALUE : FLOAT;

   begin

      -- limited random value generator to two values
      -- for debugging purposes

      if (DIRECTION_CHAR = 'X') then
         NEW_VALUE := 1.0;
      else
         NEW_VALUE := 0.0;
      end if;

      return NEW_VALUE;

   end RANDOM_VALUE;

--****************************************************************
--*                                                              *
--*          P A C K A G E   B O D Y   M I S S L E               *
--*          =============   =======   ===========               *
--*                                                              *
--****************************************************************
   package body MISSLE is

      procedure LAUNCH( MIS_X, MIS_Y, MIS_Z : out FLOAT; NUMTASK : in INTEGER ) is

      begin

      MIS_X := 528000.0;
      MIS_Y := 528000.0;
```

```
MIS_Z := 0.0;

end LAUNCH;

procedure MOVEMENT( DISTANCE, MIS_X, MIS_Y, MIS_Z : in out FLOAT; NUMTASK : in INTEGER ) is

BOM_X       : FLOAT;
BOM_Y       : FLOAT;
BOM_Z       : FLOAT;
X_DIRECTION : FLOAT;
Y_DIRECTION : FLOAT;
Z_DIRECTION : FLOAT;
X_MULT      : FLOAT;
Y_MULT      : FLOAT;
Z_MULT      : FLOAT;
DENOM       : FLOAT;
TASKPOS     : INTEGER;

begin

   TASKPOS    := NUMTASK;

   if (NUMTASK > NUMBOM) then
      TASKPOS := NUMTASK MOD NUMBOM;
   end if;

   OFFENSE(taskpos).BOMBER_POSITION( BOM_X, BOM_Y, BOM_Z );

   X_DIRECTION := BOM_X - MIS_X;
   Y_DIRECTION := BOM_Y - MIS_Y;
   Z_DIRECTION := BOM_Z - MIS_Z;

   DISTANCE := SQRT( X_DIRECTION**2 + Y_DIRECTION**2 + Z_DIRECTION**2 );

   SCREEN_IO.LOCK;
   SCREEN_IO.STRING_IO("Missle distance is              ");
   SCREEN_IO.UNLOCK;

   DENOM    := abs(X_DIRECTION) + abs(Y_DIRECTION) + abs(Z_DIRECTION);

   X_MULT   := X_DIRECTION / DENOM;
   Y_MULT   := Y_DIRECTION / DENOM;
   Z_MULT   := Z_DIRECTION / DENOM;

   MIS_X    := MIS_X + (X_MULT * MIS_MAX_SPEED);
   MIS_Y    := MIS_Y + (Y_MULT * MIS_MAX_SPEED);
   MIS_Z    := MIS_Z + (Z_MULT * MIS_MAX_SPEED);

end MOVEMENT;

procedure STATUS( ELAPSED_TIME : in DURATION; HIT_STATUS : out BOOLEAN ) is

CHANCE     :  FLOAT;
MISSLE_HIT :  BOOLEAN;

begin

   MISSLE_HIT := false;

   if (ELAPSED_TIME <= 20.0) then
      CHANCE := 0.95;
   elsif (ELAPSED_TIME < 30.0) then
         CHANCE := 0.90;
   elsif (ELAPSED_TIME < 40.0) then
         CHANCE := 0.80;
   elsif (ELAPSED_TIME < 50.0) then
```

```
              CHANCE := 0.75;
        elsif (ELAPSED_TIME < 60.0) then
              CHANCE := 0.70;
        elsif (ELAPSED_TIME < 120.0) then
              CHANCE := 0.50;
        elsif (ELAPSED_TIME < 180.0) then
              CHANCE := 0.30;
        elsif (ELAPSED_TIME >= 180.0) then
              CHANCE := 0.10;

        if (RANDOM_VALUE(X) < CHANCE) then
           MISSLE_HIT := true;

        HIT_STATUS := MISSLE_HIT;

      end STATUS;

   end MISSLE;


--*******************************************************************
--*                                                                 *
--*            P A C K A G E   B O D Y   B O M B E R                *
--*            =============   =======   ===========                *
--*                                                                 *
--*******************************************************************
   package body BOMBER is

      procedure LAUNCH( BOM_X, BOM_Y, BOM_Z : out FLOAT ) is
      begin

         BOM_X := 0.0;
         BOM_Y := 26400.0;
         BOM_Z := 0.0;

      end LAUNCH;

      procedure MOVEMENT( BOM_X, BOM_Y, BOM_Z : in out FLOAT; NUMTASK : in INTEGER ) is

      MIS_X       : FLOAT;
      MIS_Y       : FLOAT;
      MIS_Z       : FLOAT;
      X_DIRECTION : FLOAT;
      Y_DIRECTION : FLOAT;
      X_MULT      : FLOAT;
      Y_MULT      : FLOAT;
      DENOM       : FLOAT;
      DISTANCE    : FLOAT;

      begin

         DEFENSE(numtask).MISSLE_POSITION(DISTANCE,MIS_X,MIS_Y,MIS_Z);

         X_DIRECTION := RANDOM_VALUE('X');
         Y_DIRECTION := RANDOM_VALUE('Y');

         DENOM       := abs(X_DIRECTION) + abs(Y_DIRECTION);

         X_MULT      := X_DIRECTION / DENOM;
         Y_MULT      := Y_DIRECTION / DENOM;

         BOM_X       := BOM_X + (X_MULT * BOM_MAX_SPEED);

         if BOM_Y + (Y_MULT * BOM_MAX_SPEED) > 528000.0 then
            BOM_Y    := BOM_Y - (Y_MULT * BOM_MAX_SPEED);
         else
         BOM_Y    := BOM_Y + (Y_MULT * BOM_MAX_SPEED);
      end if;

    end MOVEMENT;

  end BOMBER;


--##############################################################
--#                                                            #
--#   P A C K A G E   B O D Y   P R O C E S S _ C O N T R O L  #
--#   =============   =======   ===========================    #
--#                                                            #
--##############################################################
  package body PROCESS_CONTROL is

    procedure OPENING_SCREEN is
    begin

      new_line(25);
      put_line("        Welcome to a strategic warfare defense simulation");
      new_line;
      new_line;
      new_line;
      put_line("                            created by");
      new_line;
      new_line;
      put_line("                        Jeffrey K. Lovelace");
      new_line;
      new_line;
      put_line("                      under the direction of");
      new_line;
      new_line;
      put_line("                        Dr. Albert Crawford");
      put_line("                  Department of Computer Science");
      put_line("            Southern Illinois University at Carbondale");
      put_line("                             Fall 1988");
      new_line;
      put_line("Do you need instructions on the use of this software package? (Y/N)");
      new_line;

    end OPENING_SCREEN;



    procedure DISPLAY_DIRECTIONS is

      RESPONSE  : CHARACTER;

    begin

      new_line(25);
      put_line("The software package you are about to use, simulates the event of");
      put_line("a scaled attack of nuclear bombers.  The defense against these bombers");
      put_line("comes in two forms.  One is space based High-speed Anti-Aircraft");
      put_line("Missles (HAAM) and the other is less effective conventional");
      put_line("Anti-Aircraft (AA) guns");
      new_line;
      put_line("HIT <ENTER> TO CONTINUE...");
      get(RESPONSE);

      new_line(25);
      put_line("The purpose of this simulation is to show if the HAAM defense system");
      put_line("is effective to the builders specifications before the system is actually");
      put_line("installed.  The user will be allowed to enter variables into the system");
      put_line("and then run the simulation in order to see the results.  If the results");
      put_line("meet the satisfaction of the purchaser, then the purchaser will have the");
      put_line("system installed.  Thus, the use of computer simulation is a  cost effective");
      put_line("method to see if the product performs to standards in a real time setting.");
      new_line;
      put_line("HIT <ENTER> TO CONTINUE...");
      get(RESPONSE);

      new_line(25);
      put_line("In this scaled down software package, the user will be allowed to");
      put_line("enter values for the following variables:");
      new_line;
      put_line("      1. Number of incoming nuclear bombers");
      put_line("      2. Number of defensive HAAM's available");
      new_line;
      put_line("This is a preliminary simulation in which other variables can be");
      put_line("added if desired.");
      new_line;
      put_line("HIT <ENTER> TO CONTINUE...");
      get(RESPONSE);
      new_line(25);

    end DISPLAY_DIRECTIONS;


    procedure FINAL_RESULT_DISPLAY is
    begin
      null;  -- for compilation
    end FINAL_RESULT_DISPLAY;

  end PROCESS_CONTROL;

--*****************************************************************
--*                                                               *
--*         T A S K   B O D Y   D E F E N S E _ T Y P E           *
--*                                                               *
--*****************************************************************
  task body DEFENSE_TYPE is

  START_TIME    : TIME;
  END_TIME      : TIME;
  ELAPSED_TIME  : DURATION;
  DISTANCE      : FLOAT;
  MIS_X         : FLOAT;
  MIS_Y         : FLOAT;
  MIS_Z         : FLOAT;
  NUMTASK       : INTEGER;

  begin

    NUMTASK := 0;
    DISTANCE := 728276.431;

    accept START( NUMTASK : in INTEGER );

    MISSLE.LAUNCH( MIS_X, MIS_Y, MIS_Z, NUMTASK );

    START_TIME := CLOCK;

    while ( DISTANCE > 17.32050808 ) and (MIS_X < 528000.1) loop
      MISSLE.MOVEMENT( DISTANCE, MIS_X, MIS_Y, MIS_Z, NUMTASK );
      accept MISSLE_POSITION( DISTANCE, MIS_X, MIS_Y, MIS_Z : out FLOAT );
    end loop;

    END_TIME := CLOCK;

    ELAPSED_TIME := START_TIME - END_TIME;
```

```
     MISSLE.STATUS( ELAPSED_TIME );

  end DEFENSE_TYPE;


--*****************************************************************
--*                                                               *
--*       T A S K   B O D Y   O F F E N S E _ T Y P E             *
--*                                                               *
--*****************************************************************
  task body OFFENSE_TYPE is

     NUMTASK  : INTEGER;
     BOM_X    : FLOAT;
     BOM_Y    : FLOAT;
     BOM_Z    : FLOAT;
     MIS_X    : FLOAT;
     MIS_Y    : FLOAT;
     MIS_Z    : FLOAT;
     DISTANCE : FLOAT;

  begin

     NUMTASK := 0;

     accept START( NUMTASK : in INTEGER );

     BOMBER.LAUNCH( BOM_X, BOM_Y, BOM_Z );

     accept BOMBER_POSITION( BOM_X, BOM_Y, BOM_Z : out FLOAT );

     DEFENSE(numtask).MISSLE_POSITION( DISTANCE, MIS_X, MIS_Y, MIS_Z );

     while (DISTANCE > 17.32050808) loop
        accept BOMBER_POSITION( BOM_X, BOM_Y, BOM_Z : out FLOAT );
        DEFENSE(numtask).MISSLE_POSITION( DISTANCE, MIS_X, MIS_Y, MIS_Z );
     end loop;


  end OFFENSE_TYPE;


--*****************************************************************
--*                                                               *
--*           T A S K   B O D Y   C O N T R O L                   *
--*                                                               *
--*****************************************************************
  task body CONTROL is

   RESPONSE : CHARACTER;
   BOMINDEX : INTEGER;
   MISINDEX : INTEGER;

   begin

      PROCESS_CONTROL.OPENING_SCREEN;

      get(RESPONSE);

      if (RESPONSE = 'Y') or (RESPONSE = 'y') then
         PROCESS_CONTROL.DISPLAY_DIRECTIONS;
      else
         new_line(25);
      end if;
      put_line("How many incoming nuclear bombers?");
      get(NUMBOM);
      new_line(25);

      put_line("How many defensive HAAM's are available?");
      get(NUMMIS);
      new_line(25);

      BOMINDEX := 1;
      MISINDEX := 1;

      while (BOMINDEX <= NUMBOM) or (MISINDEX <= NUMMIS) loop
         if (BOMINDEX <= NUMBOM) then
            OFFENSE(bomindex).START( bomindex );
            BOMINDEX := BOMINDEX + 1;
         end if;

         if (MISINDEX <= NUMMIS) then
            DEFENSE(misindex).START( misindex );
            MISINDEX := MISINDEX + 1;
         end if;
      end loop;

   end CONTROL;


--*****************************************************************
--*                                                               *
--*            T A S K   B O D Y   S C R E E N _ I O              *
--*                                                               *
--*****************************************************************
   task body SCREEN_IO is

   OUT_LINE : STRING30;
   OUT_INT  : INTEGER;
   OUT_REAL : FLOAT;
   PRINT_S  : STRING30;
   PRINT_I  : INTEGER;
   PRINT_R  : FLOAT;

   begin

   loop

      accept LOCK;

      loop
         select
            accept UNLOCK;
            exit;
         or
            accept STRING_IO ( OUT_LINE : STRING30 ) do
               PRINT_S := OUT_LINE;
            end STRING_IO;
            put(PRINT_S);
            new_line;
         or
            accept INT_IO ( OUT_INT : INTEGER ) do
               PRINT_I := OUT_INT;
            end INT_IO;
            put(PRINT_I);
            new_line;
         or
            accept REAL_IO ( OUT_REAL : FLOAT ) do
               PRINT_R := OUT_REAL;
            end REAL_IO;
         put(PRINT_R);
        new_line;
   end select;

   end loop;

end loop;

end SCREEN_ID;


begin
-- main program statements
null; -- for compilation
end MAIN;
```