Spring 1990

Data Communication Between an Expert System Shell and a Conventional Algorithmic Program With Application to Cam Motion Specification

Paul P. Lin
Cleveland State University, p.lin@csuohio.edu

An-Jen Yang
Cleveland State University

Publisher's Statement
The final publication is available at Springer via http://dx.doi.org/10.1007/BF01200243

Original Citation

Lin, P. P., , & Yang, A. J. (1990). Data communication between an expert system shell and a conventional algorithmic program with application to cam motion specification. Engineering with Computers, 6(2), 113 - 119. doi:10.1007/BF01200243

Data Communication Between an Expert System Shell and a Conventional Algorithmic Program with Application to Cam Motion Specification

P.P. Lin and A.J. Yang
Mechanical Engineering Department, Cleveland State University, Cleveland, OH, USA

Abstract. Although more and more expert system shells have begun to provide communication interfaces to conventional procedural languages, the interfaces are basically shell- and language-dependent. This paper presents a simple, shell- and language-independent data communication technique between a shell and a procedural language via a concept analogous to the handshake data transmission used in microprocessors. A control file is used for the action of handshake. The communication interface is between two data files in two different language environments. A program written in a LISP-based expert system shell, OPS 5, and one written in a procedural language, FORTRAN, were tested to verify the presented technique.

An expert system for cam motion specification, which needs the following three tasks—symbolic representation, numerical computation, and their communication—is described as one of the possible applications of the technique. These three tasks are essential to automated engineering design and analysis.

1 Introduction

The term "expert system" is used in artificial intelligence to refer to a computer program that performs a task in a specific domain at the level of a human expert. The primary difference between expert and nonexpert systems is that expert systems deal with knowledge processing for reasoning or inference, whereas nonexpert systems deal with numerical and nonnumerical data processing for computation or information retrieval.

The name OPS 5 stands for official production system, version 5. It is a widely used rule-based expert system shell, namely, production system. A production system is appropriate when the knowledge to be programmed naturally occurs in rule form, when a program's control is extremely complex, or when a program is expected to be significantly modified over a long period of time. Programs in a production system have complex control structures and greater flow of control variations caused by current data. In a production system more separation of knowledge (in the rules) from the control (provided by the executer) is allowed [1]. A consequence of this separation is that the executer in the production system model is a more complex object than the executer in the procedural model. A production system program may be compiled to create an efficient representation for the rules. Even if the program is compiled, it can execute only in an environment that includes a complex language executer, the inference engine.

A conventional computer program is developed by explicitly stating all applicable rules and their precise sequence of execution. The expected behavior of a conventional algorithmic program, such as in the procedural language FORTRAN, is embedded in a code. One can read the code and see the behavior. More specifically, in procedural computation of a conventional algorithmic program, knowledge about the problem domain is mixed with instructions about the flow of control. With a procedural language, a set of instructions may be compiled into an object-language program that can be executed without further need for the compiler.

The OPS 5, written in an artificial intelligence language LISP, does not provide a direct interface with a procedural language program. However, the OPS 5 does provide a mechanism, called the result element, through which working memory elements can be communicated to a LISP function. From LISP any compiled object code (including compiled FORTRAN subroutines) can be called. Such an indirect interface is very much shell-dependent, which requires the programmer to understand and follow the interface procedure carefully and correctly.

TOPSI, the personal computer version of OPS 5 written in C language uses the relocatable object files to allow users to link with external functions and call them through the normal OPS 5 external mechanism. The external.c file contains the source

Offprint requests: P.P. Lin, Mechanical Engineering Department, Cleveland State University, Cleveland, OH 44115, USA.

code for the functions that provide the interface between the TOPSI and the user-supplied external functions. After compiling the new external.c and any other files with external functions, they are linked using the batch command file. Another C-based expert system shell designed for personal computer users provides the option of source code generation in FORTRAN so that an interface to FORTRAN callable external functions becomes possible. However, the data is transmitted through an external function to the FORTRAN program one at a time.

It may be desirable for a user to run an expert system shell and a conventional algorithmic program simultaneously, and to communicate data whenever necessary. The presented communication technique is based on the concept of the handshake data transmission technique used in microprocessors. With this technique, linking two different compiled source codes and communicating data between them through external functions are not required.

2 Methodology of Data Communication

2.1 *Structure of OPS 5*

OPS 5 is more suitable to forward-chaining control flow. It contains three major components: (1) data memory or working memory, (2) production memory or rule memory, (3) an inference engine that executes the rules. The way the inference mechanism functions is that the rules in the production memory (knowledge base) are compared to elements in the data memory (data base) until a rule is instantiated and new information is placed in the latter. The iteration continues until no production rule fires.

The main advantage of using OPS 5 is its stability, efficiency, and file action capabilities [2]. The main disadvantage is its lack of a well-developed user interface. In particular, it lacks editing or explanation facilities and provides no aids for maintaining test-case libraries [1].

2.2 *Handshake Type of Data Communication*

Handshake data transmission is used in microprocessors. For instance, the 8-bit M6800 microprocessor uses a peripheral control line (CA2) in the peripheral interface adapter (PIA) [3]. This control line can be programmed to function as an interrupt input line or a peripheral output line. The function of this line is programmed with control register A. The status of bits 3, 4, and 5 of control register A determine how it is to function. Bit 5 determines whether the CA2 line is to be an input line or an output line. The handshake mode is used when a peripheral is transmitting data to the microprocessor (MPU). The peripheral must tell the MPU when it has some data, and the MPU must tell the peripheral when it has taken the data.

Analogous to the aforementioned handshake data transmission is the developed two-way data communication technique, which is described as follows. A program written in OPS 5 (program A) and one written in FORTRAN (program B) have data files A and B, respectively. A control file (file C) is used to control the action of communication. Programs A and B continue to read the updated value of *N* in file C. Here file C and the data of *N* are analogous to the control line and the data register in a microprocessor, respectively. The *N* value can be set to 1, 2, or 3 depending on what action is to be taken. Initially, *N* is set to 2. Whenever numerical computation is needed, *N* is set to 1. In this case program B (FORTRAN) is activated. After the numerical computation has been performed, *N* will be set to 2. In this case program A (OPS 5) will be activated and program B will be inactivated. If *N* has not been set to 1 yet (i.e., program B remains inactivated), the program will continue to run and check the updated value of *N*. However, the FORTRAN program will stop when the program pointer reaches the end-of-file. In order to keep the program continued to read the *N* value, the end-of-file needs to be detected first and the pointer is then instructed to go to a designated statement as soon as it reaches the end-of-file. When there is a need to terminate these two programs, *N* will be set to 3 in either program depending on the user's stopping criterion.

2.3 *Data Communication Between Two Data Files*

Data communication between two data files is described as follows. Program A writes its output to data file A and program B reads the data in data file A as the input. Likewise, program B writes its output to data file B and program A reads the data in data file B as the input. Control file C is used to tell program A when it has some data to read from data file B and to tell program B when it has some data to read from data file A. It should be noted that these two programs continue to run all the time until *N* is set to *3*. When one program is active, the other one

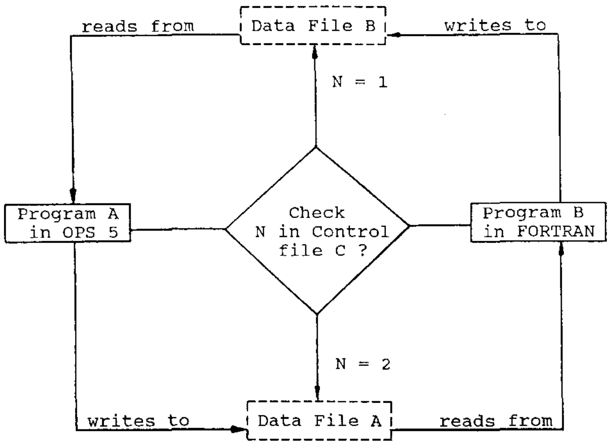


Fig. 1. Schematic diagram of data communication.

must be inactive. The inactive program does nothing but check the value of N continuously. The schematic diagram of data communication between program A (in OPS 5) and program B (in FORTRAN) is shown in Fig. 1. The flowchart of program A is shown in Fig. 2. The structure of program B is very similar to that of program A.

2.4 User Interface

In using a VAX computer (11/780) with the UNIX operating system, two different programs can open the same file without using the command SHARE. More conventionally, the user can run both programs in the same computer terminal using a multitask or multiwindow technique. For instance, the user can use the upper half of the screen (the first window) and the lower half of the screen (the second window) to monitor the results of these programs. Two small programs were written to verify the presented data communication technique. Table 1 shows the intermediate results of these two programs in a terminal screen with the use of multiwindow display. Although in these two programs only the data of a parameter was communicated, the presented technique can be used for the communication between two data files.

3 Expert System for Cam Motion Specification

3.1 Introduction to Cam Motion

Traditionally, a cam designer has to look over many standard cam motions in order to specify the segments of a complete cam rotation. The procedure

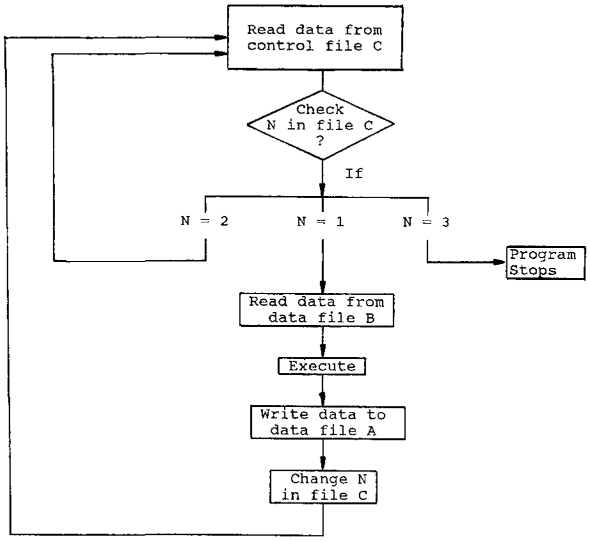

Read data from control file C
Check N in file C ?
If
N = 2
N = 1
N = 3
Program Stops
Read data from data file B
Execute
Write data to data file A
Change N in file C

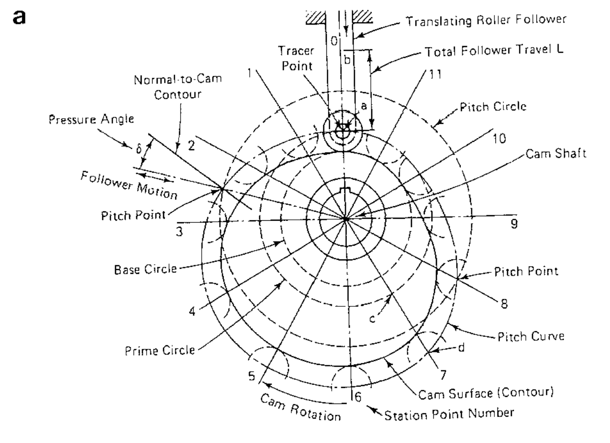
Fig. 2. Flowchart of program A.

Table 1. The result of data communication between two programs[a]

```
-1--------------------------------------------
50.  loop-action 94
51.  result-test 96 95
52.  out-put-to-screen 98

The result from the OPS 5 program is. . . . . . . . . . . 35.0
53.  starting 99
54.  loop-action 101
55.  loop-action 103
57.  out-put-to-screen 107Interrupt:
Break nil
⟨1⟩:
-2--------------------------------------------
$ fort
   The result from the FORTRAN program is . . .    6.0
   The result from the FORTRAN program is . . .   12.0
   The result from the FORTRAN program is . . .   18.0
   The result from the FORTRAN program is . . .   24.0
   The result from the FORTRAN program is . . .   30.0
   The result from the FORTRAN program is . . .   36.0
```

[a] Two test programs were written in OPS 5 and FORTRAN, respectively. The initial input for the OPS 5 to read is 0. Then 5 is added to this value each time the program is executed. Thus, the initial output is 5, which will be read by the FORTRAN program as the input. Then 1 is added to this value each time the FORTRAN program is executed. Thus, the first result from the program is 6. The process is repeated until both programs are interrupted or stopped. The intermediate results listed here are from these two programs with the use of multiwindow display. The first window shows the result from the OPS 5 after six executions. The second window shows the result from the FORTRAN program after six executions.



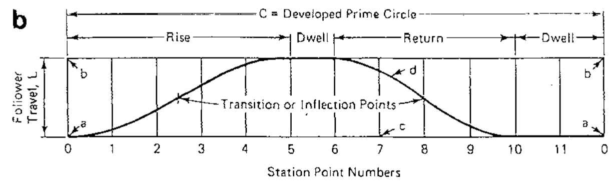


Fig. 3. (a) Disk cam and radial follower; (b) the corresponding follower profile.

may become tedious and time-consuming when there exists five or more segments within a complete rotation. Thus, it is desirable to build an expert system for cam motion specification.

A cam is a mechanical element used to drive another element, called the follower, through a specified motion by direct contact. Figure 3 shows a disk cam and the corresponding follower profile [4]. The follower motions can have almost any desired characteristics. In performing cam design, one needs to specify cam motions first, in particular, standard cam motions. They are constant velocity, parabolic, simple harmonic, modified harmonic, cycloidal, trapezoidal acceleration and modified trapezoidal acceleration, and so on [5,6]. The nonstandard approach in designing cams is to synthesize appropriate motion curves with polynomial equations. In the polynomial cam design the fifth-order polynomial (known as the 3-4-5 polynomial) and the eighth-order polynomial are commonly used.

Many equations might be used to represent the different segments of a cam's displacement diagram. The task for a cam designer to perform is to join the segments together to form the motion specification for a complete cam rotation. More specifically, derivatives of displacement diagrams need to be matched on the boundary of any two consecutive segments. In cam motion specification, lift (maximum rise) L and cam rotation angle *BETA* for each

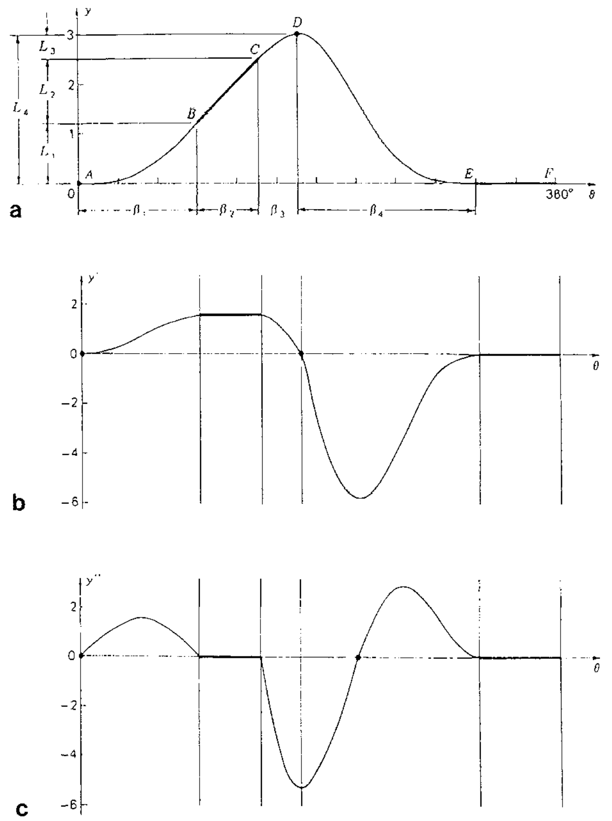


Fig. 4. (a) Displacement diagram; (b) velocity diagram; (c) acceleration diagram.

segment are to be solved. The procedure can be stated as follows [6]:

1. The motion requirements of the particular application are met.
2. The displacement, velocity, and acceleration diagrams are continuous across the boundaries of the segments.
3. The maximum magnitudes of the velocity and acceleration peaks are kept as low as possible consistent with the previous two conditions.

3.2 Illustrative Example

In order to illustrate the specification of a complete displacement diagram, an example given in Ref. [6] is used here. The problem is stated as follows.

A plate cam with a reciprocating follower is to be driven by a constant-speed motor at 150 rpm. The follower is to start from a dwell, accelerate to a uniform velocity of 25 in./sec, maintain this velocity for 1.25 in. of rise, decelerate to the top of the lift, return, and then dwell for 0.067 sec. As shown in Fig. 4, only segments *BC* and *EF* are known. In

Table 2. Characteristics of various types of cam motions

Type of motion	Equation number	Velocity factor	Acceleration factor	Cam speed application
Constant velocity	1	1.00	Infinite	Low
Parabolic	2	2.00	4.00	Medium
Simple harmonic	3	1.57	4.93	Medium
Cycloidal	4	2.00	6.28	High
Cubic curve # 1	5	3.00	12.00	Low
Cubic curve # 2	6	1.50	6.00	Low
Cubic curve # 3	7	2.00	8.00	Low
3-4-5 polynomial	8	1.88	5.77	High
Trapezoidal	9	2.00	5.33	High
Modified trapezoidal	10	2.00	4.89	High

specifying the motion for segment *AB*, the requirement is that all diagrams (displacement, velocity, and acceleration) begin with zeros at point *A* and end with nonzeros for the first two diagrams, and zero for the third diagram. It was found that only half-cycloidal rise motions can meet such a requirement. However, there are at least two possible candidates for segment *CD*. They are half-harmonic rise and half-cycloidal rise motions. In this segment the displacement diagram should start from 0 and end at a positive number. The velocity diagram should start from a positive number and end at 0. There is no restriction on the acceleration. However, the chosen acceleration will affect the selection of the next segment. For example, if half-harmonic rise motion were chosen, then the full-return modified harmonic motion may have to be used for segment *DE*.

3.3 *User-Interactive Programming*

The procedure for selecting the possible cam motions and matching the displacement, velocity, and acceleration curves is often tedious. Characteristics of various types of cam motions with the recommended cam speed applications are shown in Table 2 [7]. When using the OPS 5 program, the designer is first asked to specify the output file name, then select the cam speed. The available speeds are low, medium, and high. In high-speed applications the rate of change of acceleration that is the third derivative of the displacement (known as jerk) should be taken into account. Jerk is an indication of the impact characteristics of the loading. It may be said that impact has jerk equal to infinity [8]. Once the speed has been selected, the number of possible candidates will be reduced. Notice that parabolic and simple harmonic motions are recommended for use at medium speed. In this case velocity and acceleration factors will be compared. The designer can select a cam motion with lower peak value of velocity and higher peak value of acceleration. However, for the purpose of minimizing peak dynamic loads, it is important to minimize the peak value of acceleration. Thus, the parabolic motion will be recommended by the expert system program. The acceleration curve with abrupt changes in the parabolic motion will, however, exert abruptly changing contact stresses at the bearings and on the cam surface and lead to noise, surface wear, and eventual failure [8]. This explains why the parabolic motion should not be used for high-speed applications. The expert system at this stage will make recommendations and provide reasoning in cam motion selection to the user.

3.4 *Motion Diagram Classification*

Each diagram of a specific cam motion is designated to an equation number. Under the number, any diagram is classified as 0 to 1, 0 to -1, 1 to 0, and -1 to 0 where -1, 0, and 1 represent the low (negative), zero, and high (positive) levels, respectively. If, for instance, the displacement diagram for the second segment is 0 to 1, then the one for the first and third segments may be selected by the OPS 5 program with -1 to 0 and 1 to 0, respectively. After the appropriate equation number for each segment has been selected in the OPS 5 program, the FORTRAN program will perform numerical computation for the lift and the cam rotation angle for each segment. The computed data (in a data file) is then transmitted to the OPS 5 program.

4 Conclusions

The use of control files and data files was found to be a simple and efficient way to communicate data. The presented communication technique is analogous to the handshake data transmission technique used in microprocessors. The main advantage is that the communication interface does not require linking two different source codes and communicating data through external functions. The technique is shell- and language-independent.

Cam motion specification, an important task in cam design, is usually performed by an experienced cam designer. It is desirable to build an expert system for cam motion specification as the first step of automated cam design. The expert system will recommend the appropriate cam motion for the selected cam speed; match the motion diagrams at the connections between two adjacent segments; calculate the follower's lifts and the cam rotation angles; and finally, return to the user for making a decision on the cam motion specification.

Recently, some efforts toward automated design have been made by researchers such as Dixon and his colleagues [9] who developed expert systems for mechanical design. The term "mechanical design" is most often defined as the creative decision-making process for specifying or creating physical devices to fulfill a stated need [10]. This definition describes what mechanical design is but gives no indication as to how it is done. The actual process of mechanical design is not so well understood, even though there are a number of books on the subject [11,12]. The first phase of the design process as stated in Ref. [10] is to transform the problem into a well-formed set of design specifications. In this phase symbolic representation and numerical computation are needed in building a complete expert system for design. Of equal importance, two-way data communication technique between them can make the system more friendly and convenient to use. In particular, the user of an expert system shell does not have to exit and enter so frequently. Moreover, the user can make on-line decisions while programs are running.

References

1. Brownston, L., Farrel, R., Kant, E., Martin N (1986) Programming Expert Systems in OPS 5: An Introduction to Rule-Based Programming. Reading, PA: Addison-Wesley Publishing
2. Waldron, K.J., Waldron, M.B., Wang, M. (1986) An expert system for initial bearing selection. Presented at the Design Engineering Technical Conference in Columbus, OH. ASME Paper No. 86-DET-125
3. Bishop R. (1979) Basic Microprocessors and the 6800. Rochelle Park, NJ: Hayden Book Co. pp. 164–175
4. Erdman, A.G., Sandor, G.N. (1984) Mechanism Design: Analysis and Synthesis. Vol. 1. Englewood Cliffs, NJ: Prentice-Hall, 275–276
5. Rothbart, H.A. (1956) Cams: Design, Dynamics, and Accuracy, New York: John Wiley & Sons, pp. 182–213
6. Shigley, J.E., Uicker, J.J. (1980) Theory of Machines and Mechanisms. New York: McGraw-Hill Book Co., pp. 193–240
7. Jensen, P.W. (1987) Cam Design and Manufacture. Second Edition. New York: Mercel Dekker, Inc.
8. Mabie, H.H., Reinholtz, C.F. (1987) Mechanisms and Dynamics of Machinery. Fourth Edition. New York: John Wiley & Sons, pp. 71–127
9. Dixon J.R., Libardi E.C., Luby S.C., Vaghul, M. (1987) Expert systems for mechanical design: Examples of symbolic representations of design geometries. Eng. Comput. 2(1), 1–10
10. Ullman, D.G., Dietterich, T.A. (1987) Mechanical design methodology: Implications on future developments of computer-aided design and knowledge-based systems. Eng. Comput. 2(1), 21–29
11. Love, S.F. (1980) Planning and Creating Successful Engineered Designs. Advanced Professional Development, Inc., Los Angeles
12. Ostrofsky, B. (1977) Design, Planning, and Development Methodology. Englewood Cliffs, NJ: Prentice-Hall

Appendix—The OPS 5 and FORTRAN Programs

```
;-------------------------------------------------
;   This is the OPS 5 program designed to demonstrate the
;   data communication with a FORTRAN program
;
;   NOTE:  In this program the data received from the
;          output of the FORTRAN program is added by
;          5 each time
;
;   File designation
;                          C :  the control file for both OPS 5 and
;                               FORTRAN program
;                          C1:  the output file of the OPS 5
;                               program (for the FORTRAN
;                               program to read)
;                          C2:  the input file of the OPS 5 program
;                               (generated by the FORTRAN
;                               program)
;-------------------------------------------------
(literalize start)
(literalize loop)
(literalize result answer)
(literalize out-put-to-screen data)
; # 1.  starting : ========================
        (p starting
        (start)
     →
        (make result ^answer 1)
        (make loop) )
; # 2.  loop-action : =====================
           read data from control file C
        ============================
          (p loop-action)
          (loop)
        - (result ^answer 2)
        →
          (openfile input |c| in)
          (make result ^answer (accept input) )
          (closefile input)
          (make loop) )
; # 3.    result-test : ========================
;         read data from control file C2 (generated by the
;         Fortran program)
          ==============================
          (p result-test)
          (loop)
          { (result ^answer 2) ⟨n3⟩}
        →
          (remove ⟨n3⟩)
          (openfile date-from-fortran |c2| in)
          (make out-put-to-screen ^data (accept
          date-from-fortran) )
          (closefile date-from fortran) )
; # 4.    output-to-screen ====================
;         show the result on screen and put it into file C1,
;         update the data in control file C
;         ==============================
          (p out-put-to-screen)
          (out-put-to-screen ^data ⟨value⟩)
        →
          (bind ⟨value⟩ (compute ⟨value⟩ + 5) )
          (write (crlf) (crlf)
          The result from the OPS 5 program is . . . . . . . .
          . . ⟨value⟩)
          (openfile output |c| out)
          (write output ⟨value⟩ (crlf) )
          (closefile output)
          (openfile reope |c| out)
          (write reope 1 (crlf) )
          (closefile reope)
          (make start) )
          (make start)
          (run)


c=================================
c     This is the FORTRAN program designed to demonstrate
c     the data communication with an OPS 5 program
c
c     NOTE:  In this program the data received from the
c            output of the OPS 5 program is added by
             1 each time
c
c     File designation
c                         C : the control file for both FORTRAN
c                             and OPS 5 programs
c                         C1: the input file of the FORTRAN
c                             program (generated by the OPS 5
c                             program)
c                         C2: the output file of the FORTRAN
                              program (for the OPS 5 program
                              to read)
c=================================
      integer a
   51 open (unit=8, file='C', status='old')
      rewind 8
      read (8,*,end=31) N
      close (8)
c
c. . . . . . . . read data from the control file C
c
      if (N. eq. 1) then
      open (unit=9, file ='Cl',status='old')
      read (9,*) x
      close (9)
c. . . . . . computation example
      x=x+1
c. . . . . . the computation is now completed
c. . . . . . set the data in the control file C to 2
      write (6,*)  'The result from the FORTRAN program is
     #. . .', x
      open (unit =10, file='C2', status='old')
      write (10,*) x
      close (10)
      open (unit=8, file ='C', status='old')
      write (8,*) 2
      close (8)
      go to 51
      else if (N. eq. 2) then
c. . . use the command SLEEP for 3 seconds
c     to save CPU time (optional)
      a=sleep (3)
      go to 51
      else if (N. eq. 3) then
      go to 100
      end if
c
   31 close (8)
      go to 51
c
  100 stop
      end
```