PROCEEDINGS

OF

SHARE XLI

August 13 - 17, 1973

The Deauville Hotel

Miami Beach, Florida

SESSION REPORT

IF: An Interactive FORTRAN Compiler — Session Title

B402 — Session Number

August 15 — Date

1:30 — Time

Riviera — Room

BASIC Systems — Division

Michigan Terminal System — Project

Committee

D. A. Twyver — Chairman

UBC — Inst. Code

University of British Columbia — Affiliation

Computing Centre UBC, Vancouver, Canada (604) 228-3072 — Address and Phone

SPEAKER:

R. Hall (UBC)

ABSTRACT:

An Interactive FORTRAN compiler (IF) developed at the University of British Columbia was described. IF is a highly interactive system for the development and debugging of FORTRAN programs. It translates FORTRAN source statement by statement to an internal code which is then interpretively executed. Input may be free form and an immediate execution mode is provided. Access to the MTS editor is provided for program revision, as well as the ability to monitor and control executing programs using break-points and at-points. The system is operational under tne MTS system and a live demonstration accompanied the presentation.

The following pages contain the text of Mr. Hall's presentation, a transcription of the videotape accompanying the presentation, and a transcription of part of the question-and-answer exchange which followed the presentation.

In April 1971, we began a project at the University of British Columbia with the purpose of designing and implementing a system which would facilitate FORTRAN program development. The Interactive FORTRAN (or IF) system which emerged from this project was successfully released to general users in April of this year, just under two years from the time the project was begun.

The IF project, we feel, has been a progressive step in the direction of providing program development assistance, not only with FORTRAN, but with higher level languages in general. The background gained in the implementation of the IF system provides the basis for a uniform approach to this problem, the significance of which will increase as more and more conversational computing systems become available.

Let me briefly outline some of the design objectives of the user interface to the IF system;

The first, and probably foremost consideration was that

1. programs would be uniformly transportable between the IF system and our conventional FORTRAN batch compiler.

   (A conversational front-end for a batch compiler is a simple solution to this problem, but the following objectives make it evident that such a scheme is out of the question).

As a second objective, we proposed the IF system be comprehensive in that it would

2. provide facilities for compilation, execution, and editing. Each of these processes would be carried out under stringent error monitoring; meaningful, non-coded error diagnostics containing symbolic information would be produced in the event of an error; and all errors would be fully recoverable. In addition, the user would be given facilities for control of his program during execution.

Thirdly, the IF system would, if possible

3. present a non-complex face to the user.

In our implementation the first objective, that of program transportability, has virtually been attained. This can be attributed primarily to the fact that IF provides free access to the standard user program support; the FORTRAN I/O package, the elementary FORTRAN function library, and probably most important the general program library.

Our goal of obtaining a comprehensive program development package has also been attained. Forming the framework of IF are facilities not only for controlled execution, but for compilation and editing as well, with each operating under close scrutinization for errors. Control of the system is provided by a unique immediate execution process which will be described in a moment.

The flexibility of the IF system can be illustrated by noting that each of the traditionally disjoint development processes (compilation, execution, and editing) are allowed to overlap almost entirely at will. This, combined with the immediate execution process, means that program errors may be corrected "on the fly" -- that is, the source program may be immediately edited in the event of an error, with the corresponding changes reflected in the running program when execution is resumed. Often it is not necessary to begin all over again when an error has been encountered, but rather the user may choose to step from error to error until execution completes successfully. Of course, the user who prefers to start again from the beginning can do so with equal ease.

Any system which is proposed to be an aid to interactive debugging must have a facility for interrogating and altering the values of variables during execution. IF fulfils this criterion in a very natural, but surprisingly effective way. A running program may halt for any of several reasons -- we call this interruption of execution a "suspension", for as implied earlier, execution of the program remains pending. When a suspension is in effect, the user may enter IF commands or FORTRAN statements for immediate execution.

FORTRAN statements used in this context are very much like commands -- their function remains essentially the same with the difference that the action represented by the statement is carried out immediately. For example, the values of variables in a routine may be displayed by simply entering a list-directed PRINT statement; or values in the program may be changed by entering a FORTRAN assignment statement. This facility extends to include almost all FORTRAN statements -- CALL, READ, IF, RETURN and so on.

The benefits of such a scheme should be apparent:

1. the user works with a language with which he must already be familiar (FORTRAN) and
2. changes can easily be made to a compiled program during its execution, meaning the user does not need to include debugging statements (which generally turn out to be inappropriate or inadequate anyway) prior to compilation of his program. Instead he can adjust his debugging attack according to the currently prevailing circumstances.

The major features included to provide the user with control of execution are BREAK-points and AT-points, (which pre-specify locations in the program at which a suspension is to occur) and commands which permit execution to proceed in steps of one or more statements.

The third design objective, particularly in light of the versatility of the IF system, has been difficult to attain. It is interesting to note that features included to remove some of the tedious rigidity of programming can contribute an unwholesome amount to the apparent complexity of a system such as IF. The case in point is the provision for free-format FORTRAN -- it is the only reasonable way to program conversationally, but nonetheless it can cause difficulty for the uninitiated user.

In an effort to minimize the amount of work required on the part of the user in learning how to use the IF system, we have provided commands which closely parallel MTS commands in both function and syntax, and as mentioned earlier, immediate execution of FORTRAN statements. In addition, the MTS editor has been incorporated, (with some changes) into the IF system, which again presents a familiar facility to our MTS users.

There are a large number of commands provided by the IF system, but with just a selected few the user gains control of a versatile and powerful debugging tool. Additional commands can be introduced as his knowledge and ability to use the IF system increase.

Response to the IF system from our user community has generally been very favorable, and usage has been steadily increasing since the date of release. We anticipate that this trend will continue as more and more users become aware of the benefits that can be had by using Interactive FORTRAN.

```
1    # $run *if
     # EXECUTION BEGINS
     * IFC(UN123)
2    * $compile prey.s
     WARNING:$LIST IFC:NEWS - *IF NOW EDITS USER'S FILES (E.G; FILE "PREY.S")
     ROUTINE NAME:  MAIN
     ROUTINE NAME:  TRACE
     ROUTINE NAME:  WFILE
     * $run main
3
        AUTHOR/SUBJECT   REF.NO.                  REPRINT SOURCE

     $endfile
     STOP
     + $release
4    * $compile
         1_ 100    i=1
         2_
     ROUTINE NAME:  MAIN1
5    * $compile
         1_ m=0
     ROUTINE NAME:  MAIN2
         2_ do 10 j=1,5
6        3_ m=m+j=
     M=M+J=
           $
     ERROR:  EXTRA CHARACTERS AFTER COMPLETE STATEMENT
7    : print
     :       3           M=M+J=
     : alter 'j='j'
     :       3           M=M+J
8    : stop
         4_ 10 continue
         5_ print,m
         6_ stop
         7_ end
9    * $list main2
     *     1           M=0
     *     2           DO 10 J=1,5
     *     3           M=M+J
     *     4        10 CONTINUE
     *     5           PRINT,M
     *     6           STOP
     *     7           END
10   * $run main2
                15
     STOP
11   + $release
12   * $break main2:4
     * $run main2
     AT LINE      4     OF ROUTINE MAIN2
13   ***** BREAKPOINT
14   + print,m
                 1
15   + $step
     AT LINE      3     OF ROUTINE MAIN2
16   + $restart
     AT LINE      4     OF ROUTINE MAIN2
```



```
     ***** BREAKPOINT
17   + $remove main2:4
18   + $at main2:4
         1_ print, ' m=',m
         2_ end
     + $restart
     AT LINE      4     OF ROUTINE MAIN2
      M=              6
     AT LINE      4     OF ROUTINE MAIN2
19    M=             10
     AT LINE      4     OF ROUTINE MAIN2
      M=             15
                    15
     STOP
     + $release
     * $compose
         1_ subroutine x(a)
20   ROUTINE NAME:  X
         2_ a=clik(0)
         3_ return
         4_ end
21   * call x(y)
     AT LINE      2     OF ROUTINE X
22   ERROR:  ROUTINE CLIK IS UNDEFINED
     + $? +2
     + MSG+ERROR:  A ROUTINE MAY BE EXECUTED ONLY IF IT HAS BEEN DEFINED
23   + TO THE "IF" SYSTEM.  IT MAY BE DEFINED BY $COMPILE'ING OR
     + $COMPOSE'ING IT OR BY $LOAD'ING IT; OR, IT IS DEFINED
     + AUTOMATICALLY IF IT APPEARS IN THE LIBRARY
     + $edit x
     : line 2
     :       2           A=CLIK(0)
24   : alter 'ik'ock'
     :       2           A=CLOCK(0)
     : stop
25   + $restart
     REPEATING STATEMENT
     *y
26       23869.16
     * sqrt(y)
         154.4965
27   * x:a=4; print,x:a
          4.000000
28   * $display routines
     * "X"  RANGE=( 1 , 4 )  TYPE=SUBROUTINE  STATUS=INITIALIZED
     * "MAIN2"  RANGE=( 1 , 7 )  TYPE=MAIN  STATUS=INITIALIZED
     * "MAIN1"  RANGE=( 1 , 1 )  TYPE=MAIN
     * "WFILE"  RANGE=( 131 , 160 )  TYPE=SUBROUTINE
     * "TRACE"  RANGE=( 109 , 130 )  TYPE=SUBROUTINE
     * "MAIN"  RANGE=( 1 , 108 )  TYPE=MAIN  STATUS=INITIALIZED
29   * $explain  $load
     *       $LOAD FDNAME
     *
     *            LOAD THE ROUTINES CONTAINED AS OBJECT MODULES IN THE
     *            SPECIFIED MTS FILE.
30   * $load *time
     * call time(0)
     CLOCK  17:47:15 DATE  08-07-73
31   * $stop
     # EXECUTION TERMINATED
```

The characters *, +, and : appearing in the first column are issued by the IF system to indicate the current mode to the user. Lines entered by the user appear in lower case.

1. $RUN *IF causes execution of the IF system to begin.

2. The name PREY.S appearing as an operand on the $COMPILE command corresponds to an MTS file containing the FORTRAN source for a MAIN routine and two sub-programs. The main program is arbitrarily assigned the name MAIN.

3. Compilation and execution can be, as we see here, somewhat similar to that of a conventional compile and run.

4. Entering a $COMPILE command without operands indicates lines that are entered at the terminal are to be compiled. Compiled lines are always saved for subsequent editing.

5. Normally we prefer to enter FORTRAN lines at the terminal in free format, so we use the counterpart of the $COMPILE command $COMPOSE. The $COMPOSE command is identical to the $COMPILE command in every way, except FORTRAN lines are entered in free format and automatically converted to IBM fixed format before being saved.

6. In the event of an error we are presented with the Editor for immediate correction of the routine.

7. The automatic invocation of the Editor is signified by the colon prefix.

8. Entering the STOP command in EDIT mode indicates editing has been completed and the compilation should be resumed.

9. Having now successfully composed a routine we now request that it be listed by entering the $LIST command, specifying MAIN2 as the routine to be listed.

10. To cause execution of this routine we again use the $RUN command specifying the name of the routine in which execution is to begin. When execution flow suspends in a particular routine, that routine becomes active.

11. The $RELEASE command simply causes the active routine to return to a non-active state, and in so doing terminates execution.

    Next we look at some of those features which distinguish IF from the conventional approaches to execution of FORTRAN programs.

12. Using the $BREAK command we have set a BREAK-point at line 4 of MAIN2.

13. The execution flow arrives at line 4 and a suspension immediately occurs.

14. When a suspension has occurred we can interrogate and change the values of variables using FORTRAN statements. In this case, we use the WATFIV list-directed print statement to print the current value of M.

15. We can then use the $STEP command to step through the program one statement at a time;

16. or allow execution to continue freely using the $RESTART command.

17. BREAK-points may be removed with the $REMOVE command.

18. The $AT command is a close associate of the $BREAK command. AT points allow the operations which are to be carried out to be prespecified.

19. When an AT-point is encountered during an execution flow, control does not return to the terminal; instead the prespecified statements are executed, following which normal execution flow is automatically resumed.

20. To demonstrate error handling during execution we will now compose a short subroutine called X.

21. To invoke this routine and to pass it an argument, we simply use a FORTRAN CALL statement.

22. The execution flow suspends at line 2 of X when a reference is made to an undefined routine. Here we have several options: we can assign a value to A; define the missing routine; or change the name of the routine being referenced. We will choose the latter, but first lets get some more information about the error.

23. The $? command can be used to increase the level of explanation of the last error encountered.

24. Now we will change the name of the routine being referenced.

25. The $RESTART command causes the statement in error to be repeated. This time the statement executes successfully. The routine CLOCK is not one which we have defined, but it was found in the General Library.

26. Entering Y and SQRT(Y) causes the value of Y and its corresponding square root to be printed.

27. The statement beginning X:A... demonstrates the ability to modify and interrogate values of variables in various routines. The value 4 is assigned to variable A in routine X and then printed.

28. ROUTINES is just one of the many parameters that may be placed on the $DISPLAY command. In this case it prints out the routines which are currently defined to the IF system.

29. The $EXPLAIN command can be used to obtain lists of all IF commands, or to obtain information about individual commands. Here a brief explanation of the $LOAD command is given.

30. $LOAD *TIME causes the object program in the file *TIME to be loaded defining the entry point TIME. In this case a self-contained program was loaded for demonstration purposes. Object modules which define subroutines may be loaded also and these will interface with routines which are compiled under the IF system.

31. Finally, a $STOP command causes execution of the IF system to be terminated.

THE IMPLEMENTATION OF IF

The implementation of the IF system was carried out using System/360 assembler language under the Michigan Terminal System at UBC (model 67 duplex). In its present form, IF represents an investment of 8 man-years and approximately one-quarter million dollars (with computing charges contributing to roughly one-half).

We had an acceptable, but restrictive, program development system going at the end of the first year. The second year was used to implement the more esoteric features, finish off some of the rough edges, and to bring the system to a degree of reliability which would be considered satisfactory to our users.

Although each of the components has the ability to produce specific debug information, the development of the system was particularly assisted by a general internal debug monitor which was developed in parallel with the overall implementation. The debug monitor which initially allowed displaying and altering of locations, later provided BREAK-points, AT-points and instruction stepping facilities for testing the IF system itself. It now includes such functions as allowing a newly developed IF module to be loaded and interfaced with the shared-memory version of IF for testing purposes, and can graphically illustrate where CPU time is being used within the IF system.

QUESTION-ANSWER EXCHANGE

Is the editor in the IF system as powerful as the CMS editor?

Answer --

The editor used in the IF system is the same as the editor which handles all conversational editing for the MTS system, a reasonably powerful editor. When running under the IF system, the user is concerned with editing a particular type of file -- namely one which contains FORTRAN source, and as such, a special set of editing functions or capabilities (available with neither the MTS or CMS editors) is desirable. We have found however, that the MTS editor is more than satisfactory for our purposes, and it has the added advantage that it does not require the MTS user to learn a new editor in order to use the IF system.

Is there any limitation on the size of a program which may be run under the IF system?

Answer --

As mentioned in the talk, programs are completely transportable between our conventional compiler and the IF system. The limitations on the size of programs which may be run is essentially the same.

A difference between WATFIV and the IF system appears to be that WATFIV very rigorously checks for program consistency, while the IF system is geared to assisting in the location of errors in program logic.

Answer --

It is true that IF is very beneficial in locating errors in program logic, but IF also provides rigorous checking of program consistency which is equal to that performed by WATFIV.

Do you support file operations from within IF and how do you communicate to IF which files you want to attach?

Answer --

IF provides all facilities which the user has available when running in the conventional manner. The user indicates which files are to be attached by specifying the correspondence on the IF $RUN command, exactly the same as with a conventional run under MTS.

Is IF an integral part of MTS or is it a separate entity?

Answer --

IF is completely separate from the MTS system. IF does, however, rely heavily on support provided by the MTS system: direct access files, program interrupt and attention interrupt control, and so on.

What is the overhead in running under IF?

Answer --

IF executes approximately 10 times slower than WATFIV. We attribute this, to a large extent, to the flexibility in the IF system. Also, the IF system was implemented with a primary goal of getting the system working and to get it working correctly. Now that this goal has been achieved, we will investigate the possibilities for speeding up interpretation. In any event, we feel that slow execution is well worth the trade-off for the excellent debugging facilities.

How do you distinguish between FORTRAN statement labels and MTS line numbers?

Answer --

IF commands which accept an MTS line number or line range will, in general, accept a statement label or statement label range as well. Statement labels are indicated by appearing on a command with a hash-mark (#) prefix.

e.g. 20.25 is an MTS line number
#100 is a FORTRAN statement label.

Can you give a line number (as opposed to a statement label) on a GO TO statement and cause execution to continue?

Answer --

No, because this would require a change to both the syntax and semantics of the FORTRAN GOTO statement, a change which we felt was inappropriate. An IF command, $RESTART, which allows either a line number or a statement label to be specified is provided for the purpose of restarting execution.

Have you run across any class of bugs which IF does not help you to find?

Answer --

The facilities provided by IF are sufficient to provide at least some assistance in locating any type of FORTRAN bug. Areas in which specific help is not given currently are program loops and program event conditions (e.g. when I becomes equal to 1000), although future implementation will likely include assistance in these areas also.

Have you experienced system degradation when running IF?

Answer --

Not a significant amount. The IF system, while being very large, does reside in shared memory which means several users use the same copy. The individual user's memory requirement with IF is dependent on the size of his FORTRAN program, and it is expected that a number of users with large FORTRAN programs could cause a noticeable system degradation.

Why has response to IF been slow?

Answer --

It's not clear how fast we should be expecting IF to catch on. IF is still new to our user's and IF does require a certain amount of familiarization before a user feels comfortable with it.

Usage statistics indicate that approximately 10% of our users are using IF and use counts indicate that IF is taking over a proportional amount of the WATFIV usage.

How does one go about getting IF and how much does it cost?

Answer --

IF is currently available only to installations running under MTS. Should a conversion to another system be carried out, we would expect the IF system to be available to non-commercial organizations at no charge.

Have you plans for running under any other system?

Answer --

The implementation of IF was carried out with some consideration to transportability to other systems, but not so much as to compromise the MTS version. While it would be possible to convert IF to run under VS2, TSS, or VM/370, UBC has no current plans to carry out such conversions.

How difficult would it be to do the conversion?

Answer --

A proper conversion would take between 4 and 12 man-months. The length of time required to do the conversion would depend heavily on the facilities and services provided by target system.

If it was possible to implement IF under TSO taking a core commitment of 200 K or less, we would be interested in having IF.

Answer --

Due to the size of the IF system (approx. 250 K) we think it would be unreasonable to attempt to use the IF system on anything but a virtual storage system. An overlay structure would be an alternative, but this would significantly increase the amount of work required for conversion.

Would it be possible to look at the structure of IF to consider the conversion of IF ourselves?

Answer --

A policy on a request such as this has not yet been established, but we expect that this would be possible.

Enquiries regarding the IF system should be directed to:

Ronald H. Hall
Computing Centre
Civil Engineering Building
University of British Columbia
VANCOUVER, Canada

S E S S I O N R E P O R T

PROJECT WORKING SESSION (Model 44 Project) — Session Title
B501 — Session Number

August 14, 1973 — Date
10:30 - 5:00 — Time
Crown — Room

BASIC SYSTEMS — Division
Model 44 Project — Project
Committee

John B. O'Loughlin — Chairman
WSA — Inst. Code
Affiliation

Digital Computing Center, P.O. Box 98, Wichita State Univ., Wichita, Kansas 67208 — Address and Phone
(316) 689-3630

The project reviewed installation reports, add-on core, and review of project library software to be placed in SPLA. A visit to the Boca Raton IBM plant for project members, a Sensor Based System onSystem/7, and a session on Graphics and Film Making on a S360/44 were sponsored by the project. The project will discuss the formation of a Sensor Based Systems Project at SHARE HOUSTON.

Attendance:

Name	Installation	Code
John Y. Ho	Night Vision Labs	NVL
M. (Penny) Baggett	Purdue High Energy Physics Lab	PDP
Steve Wineteer	HQ Strategic Air Command	SAC
John Golini	New Mexico Tech	NMI
John Moulton	Clarkson College	CCT
Joe Furnish	Cessna Aircraft	CA
R.L. Jones	Univ. of Maryland	UMH
George Gorsline, Jr.	Oberlin College	OBC
John H. Barron	IBM	IBM
Bernie Schwartz	City Planning New York City	
James McBride	University of Toronto	TY
Javier Albarran	Univ. of Maryland	UMH
David H. Lowell	Univ. of Houston	UH
Ted LoPresti	Univ. of Maryland	UMH
Richard L. Davis	Ohio University	OU
Jerry Michiner	Univ. of Maryland	UMH
John B. O'Loughlin	Wichita State University	WSA