Behavior Research Methods & Instrumentation
1982, Vol. 14(3), 363-366

CHANGR: A program for ordering FORTRAN statement numbers on a PDP-11 system

F. M. MARCHETTI
Queen's University, Kingston, Ontario K7L 3N6, Canada

Since the development of ALGOL as an alternative to FORTRAN in the early 1960s, programmers have been urged to use a structured programming style. Structure assists not only in the writing phase of the program's development but also during subsequent additions or corrections, and it provides a means of minimizing idiosyncratic styles. A program's organization should reflect both the programmer's concept of the flow of control and his representation of the problem to be solved. Such an approach provides two major assets: (1) more efficient algorithms, because the clarity enables a programmer to discover a clean and logical approach, and (2) community programs, to which corrections and additions can be provided by programmers other than the original author (Conway & Gries, 1975). Structure is inherent in languages such as ALGOL, PASCAL, and PL/I. Although it does not readily lend itself to a structured programming style, FORTRAN remains an exceedingly popular language for many applications and, along with BASIC, appears to be the main language for laboratory computer use.

Extensive use of subprograms, logical IFs, and pseudo-block structure (rather than an assortment of unconditional jumps) provides some structure to FORTRAN. Nevertheless, there is still an uncomfortable component in FORTRAN: the statement numbers. The statement numbers are not required to bear any numerical relationship to each other or to the logical flow of the program; they are merely labels. As a result, the statement labels may appear confusing, and modifications to a program are awkward, in part, because one has to keep track of the numbers not in use. One solution is to arrange statement numbers in ascending order within each subprogram. Once rearranged, it is usually clear which statement numbers are still available, and the numerical value of the label indicates part of the logical flow. Only the best-planned programs have statement numbers that increase in an orderly fashion as one proceeds from top to bottom of each routine, and even such programs tend to become scrambled when major changes are introduced. For FORTRAN programs written on large machines, there are several utilities, including TIDY (Murphy, Note 1), CLEAN (McNally & Dodson, Note 2) for IBM users, and CHANGER (Marchetti, Note 3) for Burroughs programmers, to order the statement numbers. CHANGR was written to provide the same service for FORTRAN users on PDP-11s or their equivalent.

The development of CHANGR was supported by grants from the Natural Sciences and Engineering Research Council of Canada to D. J. K. Mewhort (A0318 and E6339) and from Queen's University QUEST funds to the Department of Psychology. The author was supported by an NSERC postgraduate fellowship and an Ontario Graduate Scholarship. I thank D. J. K. Mewhort and R. von Königslöw for their advice, encouragement, and editorial acumen.

CHANGR is a FORTRAN program that reads in an RT-11 file containing one or more routines and creates a new file with statement numbers increasing from top to bottom within each routine. The present limitation is a total of 200 statement numbers, regardless of where they are found within a routine. The limitation arises because the present version is designed to run on a minimal PDP-11 configuration (e.g., a PDP-11/03 with 64 KB). Thus, CHANGR does not make use of extended memory options available on some PDP-11 systems. Using the options increases greatly the number of statement labels that CHANGR can handle within a subprogram.

Running CHANGR. The user engages in a dialogue to provide CHANGR with file names and parameter values. CHANGR requires two file names: one for the original file (i.e., the file with the mixed-up statement numbers) and a new one for the renumbered version of the original file. Keeping the names separate prevents accidental destruction of the original file.

CHANGR provides two independent means for influencing the kind of structure it imposes on a FORTRAN program. The first concerns the numerical labeling of each statement number within a subprogram; the second controls the structure across subprograms. In particular, the user supplies a base value that will be the starting statement number of the first routine. In addition, he provides an increment that determines the numerical value of subsequent statement numbers. These two numerical entries determine the structure of the statement labels within a subprogram. For example, if the base value is 1000 and the increment size is 10, the statement numbers of the first routine will begin at 1000, followed by 1010, 1020, 1030, and so on.

Finally, the user controls the structure across the subprograms by entering a constant to separate the statement numbers across subprogram boundaries. In effect, the statement numbers within a subprogram can be used as numerical indicators of a unique segment of coding. Extending the example, if a nonzero value is supplied as the constant (e.g., 1000), the second subprogram to contain statement numbers will have a starting value of the base plus the constant; for example, 1000 (the base) plus 1000 (the constant) will cause the second subprogram to begin at 2000 followed by 2010, 2020, 2030, and so on, regardless of where (numerically) the previous subprogram ended. Similarly, the third routine would start at 3000 (the updated base plus the constant) and continue with 3010, 3020, 3030, and so

0005-7878/82/030363-04$00.65/0

on. Such a procedure forces additional structure onto the appearance of the program, especially if the routines are physically connected in the output listing. If the feature is not wanted, a value of zero may be entered and, following the example, all statement numbers will begin at 1000 and be incremented by 10 within each subprogram. Caution must be exercised to ensure that the largest statement number to be created is not greater than 32767 (i.e., the largest integer that the PDP-11 FORTRAN can address).

CHANGR also enquires about optional output. Because CHANGR does not alter statement numbers that may be found in COMMENT statements, the user may request a printout of CHANGR's internal table containing the statement numbers and their locations. The information is useful if one wants to match the new statement numbers with old ones that appear in COMMENT statements. If the table output is requested, CHANGR asks whether the table should be sent to the line printer or to the console.

When the dialogue has been completed, CHANGR prints a "Running" message and begins. CHANGR is a two-pass modifier; as it completes each subprogram, it types to the console summary information about the first pass. The summary information includes the number of records in the subprogram and the number of statement numbers found in Columns 1-5 and in Columns 7-72. CHANGR then states that it is running the second pass and, when completed, informs the user of the number of records in the new subprogram.

If CHANGR finds a statement number referenced in Columns 7-72 that did not have a labeled complement in Columns 1-5, it prints a diagnostic warning that the subprogram will not compile. When any such offending FORTRAN statement is found, CHANGR writes out a warning and echos the original line to the console, but it prints a zero as the "appropriate" statement number in the new file. Thus, after CHANGR has completed its work, one may use an editor to search for a zero following a specific FORTRAN command (e.g., GO TO 0). At present, however, CHANGR does not discover redundant (i.e., not referenced) statement numbers in Columns 1-5, because a special effort is required to test for the condition.

Details. On the first pass, CHANGR inputs, line by line, the user's original file and ignores all lines with a "C" in Column 1. When a non-COMMENT line is found, CHANGR examines Columns 1-5 for statement numbers and then Columns 7-72 for other statement numbers. When a statement number is found, appropriate cells in a table are filled with information about what the number is and in which column it is located in the record. (The table is the limiting factor in the distributed version of CHANGR, and users may wish to extend the second dimension from 200 and 400 to some larger number for arrays TABLE and TABLE2, respectively.) CHANGR recognizes the standard set (and some of the extended set) of RT-11 FORTRAN commands. A listing of the statement types that CHANGR recognizes is provided in Table 1.

The MAIN routine directs all control and user interaction across both passes. During the first pass, MAIN calls routine PARTWO (Part 2), which is responsible for reading records and extracting statement numbers from Columns 1-5. In addition, PARTWO screens for the set of FORTRAN commands. When a command is encountered, PARTWO releases control to a routine specialized for each command type, and it, in turn, searches the record for statement numbers and, when necessary, reads a new record looking for continuation. When an END statement is found or an end-of-file (E-O-F) condition occurs, control is returned to MAIN, which then directs control dependent on the condition of exit: If the exit was due to an E-O-F (and no trailing COMMENT statements exist between the previous END statement and the E-O-F), the new file is saved and CHANGR stops. If COMMENTs exist, these are written out before the normal exit.

When an END statement is encountered, MAIN calls on two routines, NEWNM1 (new Number 1) and NEWNM2, to organize the contents of the internal TABLE array, that is, to place the appropriate base and increment values instead of the old ones in Columns 1-5 (NEWNM1). NEWNM2 enters the new values at the appropriate level in the table (i.e., matched according to the old values found in Columns 1-5) for the corresponding values from Columns 7-72. If the counters go beyond the TABLE dimension limits, a warning message is printed and the process is aborted.

Once the new numbers have been entered, control is passed by MAIN to routine RITOUT (write out), which provides the summary information about the run and which prints the internal table, if requested. MAIN then calls routine RANK, which takes all the relevant information from the array TABLE and appends it to array TABLE2 as one long set of information. Accordingly, TABLE2's second dimension must be at least twice the size of TABLE's second dimension (e.g., 200 vs. 400). RANK then sorts everything in the array TABLE2 according to the line number on which any statement number was found. The sort results in a simple means of flagging the record changes that must be made when the new file is being created.

Once TABLE2 has been sorted, MAIN calls routine PRTFIL (print file), which creates the new file. PRTFIL takes the statement numbers and left-justifies them (in A1 format) and stores those in array TABLE, which is being reused. After BACKSPACing the appropriate number of records,[1] PRTFIL reads a record from the original file and checks whether the record number is one for which changes are required. If no changes are required, a copy of the record is written into the new file. However, to minimize the size of the new file by exploiting variable record lengths, before writing out the

Table 1
FORTRAN Command Set Recognized by CHANGR

Command's Basic Form	Extended Form 1	Extended Form 2
	Input/Output Commands	
ACCEPT *, list	ACCEPT f, list	
DECODE(i,f,v)	DECODE(i,f,v,ERR=s)	
ENCODE(i,f,v)	ENCODE(i,f,v,ERR=s)	
PRINT *, list	PRINT f, list	
READ *, list	READ(u,*,ERR=s)	READ(u,*,ERR=s,END=s)
READ f, list	READ(u,f,ERR=s)	READ(u,f,ERR=s,END=s)
TYPE *, list	TYPE f, list	
	WRITE(u, ERR=s)	
WRITE(u,*)	WRITE(u,*,ERR=s)	
WRITE(u,f)	WRITE(u,f,ERR=s)	
	Auxiliary Input/Output Commands	
BACKSPACE u	BACKSPACE(u,ERR=s)	
CLOSE(UNIT=u)	CLOSE(UNIT=u,ERR=s)	
DELETE(u)	DELETE(u,ERR=s)	
ENDFILE u	ENDFILE(u,ERR=s)	
OPEN(p[,p] . . .)	OPEN(p,ERR=s[,p] . . .)	
REWIND u	REWIND(u,ERR=s)	
REWRITE(u)	REWRITE(u,ERR=s)	REWRITE(u,f,ERR=s)
UNLOCK u	UNLOCK(u,ERR=s)	
	Control Commands	
ASSIGN s to v		
CALL sub	CALL sub(p[,p] ,v[,v] ,&s)	
DO s I=i,v		
END		
GO TO s	GO TO (s[,s])	
IF(i)s,s,s		
IF(l) p		

*Note–Definitions: i = integer expression, l = logical expression, p = parameter (or parameter expression), v = variable name, f = format number (changed by CHANGR), s = statement label (changed by CHANGR), u = logical unit number, * = formated I/O expression, list = I/O variable list, sub = name of a subroutine: [] encloses an optional parameter or expression.*

new record, PRTFIL calls IBLNK (blank), which determines the number of characters on the line about to be written.[2]

When a line is found that requires replacement of statement numbers, PRTFIL modifies the line accordingly. To modify the line, PRTFIL calls routine GETSIZ (get the digit's size) to determine the number of characters a digit contains. GETSIZ detects a zero entry for a statement number in Columns 7-72 of the new file and warns the user that the new subroutine will not compile, but it does not abort the run. PRTFIL may find that the complete original record will not fit in the new form on a single record. Such a condition occurs when the old version of the record is close to Column 72 and contains statement numbers that are few characters in size but are to be replaced by numbers of greater width. Under such conditions, PRTFIL backtracks from Column 72 to an appropriate breaking point (presently defined as a blank, a comma, or a right parenthesis), writes up to the break point, and creates a new line by forcing an asterisk in Column 6 (used to indicate continuation to the compiler) and by placing the remainder of the old record on the new line, starting in Column 9.

PRTFIL also checks for an ampersand. This symbol is used by some compilers in a CALL statement to indicate a "computed GO TO," which allows a RETURN statement in a subroutine to denote a normal (fall through) RETURN or to act as an exit selector (e.g., RETURN 2) directing control to a statement number in the calling routine. Because of the status of the ampersand when associated with a statement number, CHANGR always ensures that the ampersand and statement label are together, even though some compilers (e.g., Burroughs) allow the two to be split across a continuation line.

Throughout CHANGR's run, any conditions defined as fatal cause an exit via routine CRASH. CRASH types the calling routine's name, types some pointers, and preserves the new file, thus allowing a direct examination of the error condition. Such an examination has proved helpful in determining whether an exit was due to a "bug" in CHANGR or to something unusual in the original file.

Programs run through CHANGR appear well planned and the coding structure is more self-evident than it is in programs that have evolved without the benefit of periodic updating of the statement labels. A natural consequence is that errors in logic (not simply in syntax) are discovered more readily in the updated version of a program than in the original. Further, by having each

subroutine begin with a unique statement label, structure is imposed on the overall program, not only on the individual routines.

Availability. CHANGR is available from the author in source form. The set of routines involves 20 separate subprograms, totaling about 2,000 lines of source code, including comments. The source files require about 130 blocks in standard RT-11 format. Each routine must be compiled and the set linked together. Accompanying the CHANGR routines is an RT-11 command file that contains the overlay structure to be specified at LINKing time.

REFERENCE NOTES

1. Murphy, H. M. *TIDY.* Unpublished manuscript, Air Force Weapons Laboratory, Kirtland Air Force Base, New Mexico, 1966.

2. McNally, S., & Dodson, D. S. *CLEAN: An adaptation of TIDY.* Unpublished manuscript, Purdue University Computing Center, 1971.

3. Marchetti, F. M. *CHANGER: A programme for ordering FORTRAN statement numbers on a Burroughs B6700.* Unpublished manuscript, Queen's University at Kingston, 1978.

REFERENCE

CONWAY, R., & GRIES, D. *An introduction to programming* (2nd ed.). Cambridge, Mass: Winthrop, 1975.

NOTES

1. Although the REWIND command would be more efficient, it does not provide the flexibility desired. Specifically, the REWIND statement would cause rereading from the beginning of the file; this is fine for the first–or only–routine in the file. However, because I envisage several routines appended together in a single file and CHANGR is routine oriented, not file oriented, I use the BACKSPACE statement to ensure that CHANGR remains within the "window" of the subroutine it is working on.

2. The IBLNK routine presently aborts (via CRASH) if an entire line is blank. This may be a nuisance for those who have compilers that fully ignore blank lines. If so, IBLNK should be modified to set the variable ISTP to 1 and RETURN instead of calling CRASH. The reason for forcing the call to CRASH is that if a compiler behaves unpredictably when confronted with a blank line (rather than give a compiler error), a potentially dangerous situation could arise, which I chose not to allow.

(Accepted for publication February 1, 1982.)