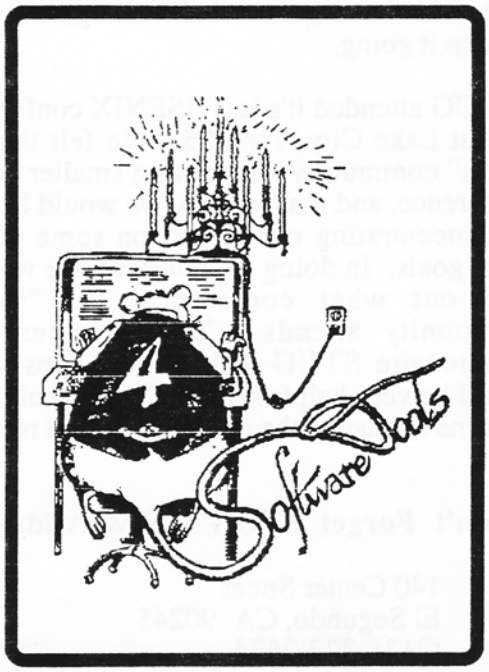
Software Tools COMMUNICATIONS

NUMBER 14 **JAN 1986**

In this Issue

As usual, we encourage you to participate in making this newsletter an active one. We have several columns you can contribute to, as well as submitting your own articles.

-- the STUG Newsletter Editor



Editor's Notes

Once again we're back on the road with a newsletter. This time we hope to make it a quarterly (or bi-monthly newsletter). But in order to really do so, we need to hear from you!

We've started a few new columns to make it easier for you to decide what to send us. You can send us questions or hints regarding shell scripts for **Scripts Corner**, or information and problems with certain tape distributions for **Tape Info**. We will even be starting a **Technical Notes** column for those who wish to submit more technical questions or hints regarding the tools. If you're looking for certain tools, or a certain distribution, let us know and we'll print it in a column called **Finder**. You can even just send a letter discussing what you've done with the tools and we'll publish it.

There have been only a few changes since the last newsletter (Nov 84, #13) was published by LBL. For starters, as many of you are aware, LBL is no longer supporting STUG (and thus no longer publishing the newsletter for us). It took us awhile to get started in the publishing business, as well as to find the time to sit down and write some of these columns. Many of the members helped by contributing articles, and even volunteering to do a column. Now that we've got the ball rolling, let's help to keep it going.

STUG attended it's last USENIX conference in Salt Lake City, June 85. We felt that the "tools" community was getting smaller at this conference, and that for now we would benefit by concentrating our efforts on some of our other goals. In doing so, however, we want to find out what conferences he "tools" community attends. This is where the questionaire **STUG Asks You** comes in. It would be very helpful for us if you would take the time to answer the questionaire and return it to us.

Don't Forget STUG's New Address:

140 Center Street
El Segundo, CA 90245
(213) 322-2574

Notice:

Since there has been such a delay in the publication of the newsletter, we will be extending all of the memberships so that individuals will receive 3 newsletters before their subscription expires. Also, please note that the expiration date is marked on you mailing label as **Month Year** after your name.

Finally, STUG has purchased an Apple Macintosh computer to keep track of the membership database, publish the newsletters, and do miscellaneous correspondence. It has proved to be very valuable.

Ongoing projects:

We are still planning on a Standards release for the primitives. In early 1986 we will be sending out the final ballot to the standards committee for review and vote. Once we receive the votes, a Standards will be published.

Just In Case You Missed An Issue:

I have heard that a few members did not receive newsletter #13. If this is also the case with you, just let us know and we will send you an issue. Also, does anyone have a copy of Issue No. 1? We do not have a copy in the office. If you have this issue, could you send us a copy. Thanks.

Issues

Vol. 1, No. 1			
Vol. 1, No. 2	Nov. 79		
Vol. 2, No. 1	Apr. 80		
No. 4	Oct. 80	No. 9	Nov. 82
No. 5	Apr. 81	No. 10	May 83
No. 6	Aug. 81	No. 11	Dec. 83
No. 7	Nov. 81	No. 12	May 84
No. 8	May 82	No. 13	Nov. 84

What's New

Hewlett-Packard Donation

In February of this year, Les Copari from Hewlett Packard had a conversation with us regarding what he might be able to do to help STUG. We discussd with him the possibilities of doing a column for the newsletter, as well as what STUG's future hopes and plans were: to standardize on the primitives, re-do the BASIC tape, have a local computer system for people to call in (Rats Nest), have a local computer system for us to be able to duplicate our tapes ourselves, etc...

Well, Les did agree to do a column about shell scripts (one of the most valuable capabilities of the Software Tools environment), but he did more than that. He looked into the possibility of having his company donate us a computer. He discovered that there was an organization in his company doing this sort of thing, he wrote a proposal for us and submitted it, and within a few weeks we received a phone call notifying us that they accepted his proposal.

We hope to find a location to house the computer system (an HP3000) and take delivery of the system later this year. Our plans at this time are to use the system for tape duplication.

We'd like to take the time to thank Les for his interest and motivation toward helping STUG with its needs.

Carousel Microtools

Carousel Microtools is a company which has developed the Software Tools for CP/M and MS-DOS systems. Unfortunately, they have recently closed their doors, but they are selling source code licenses to other businesses.

We have purchased their tools, and are offering both distributions. Each distribution includes the binary code (only, no source) for the primitives, and the source code for the public domain tools. Both distributions are avilable now for the same price as our other distributions. Carousel also have a very nice manual which we will sell separately.

If you wish to utilize *ratfor* , or if you wish to re-compile the public domain tools you will need the Microsoft FORTRAN compiler for CP/M, and the IBM FORTRAN compiler for MS-DOS.

Ratfor to C Translator

Many of you are interested in the Software Tools in C. There have been two articles in previous Communications (ref: Nov. 82 #9, Nov. 84 #13) which discuss the goals of translating or re-writing the tools in *C* . Now there is a tool which will help get everyone started with the translation process.

We feel that there are many members of our group interested in the tools in C, so we have agreeded to allow Clyde Lightfoot to distribute information about his software to the group. We hope this will be of value to. Clyde has also submitted an article discussing his software, which is published in this issue.

Software Tools Catalogue

There is a new version of the Software Tools Catalogue published by and available from Intelligent Decisions, Inc. This catalogue describes the work which has been done with the Software Tools on various systems, as well as new tools which are available for many systems.

Many of our members are contributors to the Catalogue. If you have done work with the tools or implemented new tools for the system you use you may be interested in contributing the information about what you have done for the next edition of the Catalogue. If you are looking for an implementaion of the tools on a particular system, or if you are looking for tools to do a certain task, the Catalogue might help you locate someone who can help you.

Intelligent Decisions, Inc.
P.O. Box 50174
Palo Alto, Ca. 94303
price: $8.95

Using the Tools

Letter from Ken Yap
Graduate Student
Dept. of Computer Science
University of Rochester
Rochester, NY 14627

I would like to take this opportunity to introduce myself. I am currently a graduate student at the U. of Rochester. I first heard of Software Tools back in 1976, while working on a vacation job as a programmer. Of course the source in those days came from the K&P book. The software wizard of the company I worked for had ordered the source from Addison Wesley in the form of cards.

I followed the development of the tools effort for several years and implemented the preprocessor on several machines, including PDP-11, Univac and CDC Cyber. Most of this work was done on the side for fun as an exercise in learning about different operating systems and issues of program portability.

Only in 1983 did I have a chance to use Software Tools in a couple of significant projects. By this time I had the first release of the Software Tools tape. I would like to relate my experiences below and share them with readers of the newsletter.

Sedit as a COBOL to COBOL converter

I was given the assignment of automating the conversion of a suite of about 100 COBOL programs running on a NCR to run on a VAX under VMS. I hit upon the idea of using *sedit* to automate the editing process. One feature that I added to *sedit* was to be able to chain to another script file. Without this feature, execution times would hae been too long because a single script would contain too many patterns. With chained scripts, I was able to write short scripts for the various sections of a COBOL program. The resulting scripts converted about 3 lines of source per second, but this was acceptable as manual editing required to complete the conversion tool far slower. About 90% of the differences were handled automatically and most of theremaining 10% involved some programmer thought anyway.

Telex message switching system in Ratfor

For another project I had to write a telex message switching system to accept, store and forward telex bulletins on the WMO weather information network. The host machine was to be a Perkin-Elmer 3200. For those who are not familiar with this machine, the programmer tools are to say the least, spartan. It does have a highly optimized Fortran compiler though.

I was not using the STUG preprocessor but a K&P derivative that I had hacked to generate Fortran-77 code. The final system comprised about 2000 lines of Ratfor source, not including the preprocessor itself. Using Ratfor to write the system resulted in a body of code that was easy to understand and modify. As an example of the ease of modification, generating another instance of a listener task merely involving copying a file, changing one defined symbol, and compiling. Using the include facility made it easy to share common routines and ensure that all programs using a routine had the correct version. In fact, I never had to learn how to struggle with the object librarian.

Tape Info

This column is intended for problems, questions or hints & kinks about any of the tapes which we distribute. Hopefully this column will be a way in which we communicate and possibly help eachother out with our problems. Here are a few submissions for this month:

RSX-11M Systems

The tools won't work properly if RMS is in [SYSLIB].

VAX/VMS V4.x Systems

VMS tools which run under VMS V4.x should be released at the Fall DECUS symposium via the new Languages and Tools SIG Tape. This tape will also be avialble through STUG.

Scripts Corner

Les Kopari
Hewlett-Packard Cupertino, CA

In any endeavor, it is good to set one's goals clearly and communicate them to those who may need help in meeting them. This is what this column is about. First, we'll discuss the goal. Then we'll give an example of meeting the goal. Lastly, we'll ask for your participation.

This column is a scripts corner. This is the place to send a Software Tools shell script that you have worked out, to share with others. This would make it available for review and adoption by others in the Software Tools community. Our goal, then, is to provide the vehicle for this communication.

But why should this be necessary, you may ask? Scripts are pretty trivial, obvious, simple things, aren't they? Well, that's the catch: what may seem simple to one person may be quite insightful to another. It may represent an entirely new way of thinking of a commonly known tool. In this case there is much to be gained by one's contribution. And this case may occur often enough in a broad spectrum of viewpoints.

So let's start the column off right by including a shell script for your consideration. This one is written for the revised shell, rsh, of the Software Tools for the HP3000, implemented by Ken Poulton of Terminal Software in Palo Alto, California.

First, a little packground. There is a utility program which produces a listing for each file requested. The listing contains pieces of information such as filename, creator, and date last modified. It looks like this:

```
run listdir5.pub.sys

 LISTDIR5 G.00.00 (C) HEWLETT-PACKARD CO., 1983
TYPE 'HELP' FOR AID

>listf @a;pass
********************
FILE: LISTA.TEST.AMDRD

FCODE:0  FOPTIONS: STD, ASCII, VARIABLE
BLK FACTOR: 1  CREATOR: MGR
REC SIZE: 1276 (B)  LOCKWORD:
BLK SIZE: 640 (W)  SECURITY--READ:  ANY
EXT SIZE: 5 (S)    WRITE:  ANY
# REC: 35    APPEND:  ANY
# SEC: 10    LOCK:  ANY
# EXT: 2    EXECUTE:  ANY
MAX REC: 1         **SECURITY IS ON
MAX EXT: 2  COLD LOAD ID: %21364
# LABELS: 0  CREATED: WED, 13 FEB 1985
MAX LABELS: 0  MODIFIED: WED, 13 FEB 1985
DISC DEV #: 1  ACCESSED: WED, 13 FEB 1985
DISC TYPE: 3  LABEL ADR: %156445
DISC SUBTYPE: 0  SEC OFFSET: %5
CLASS: DISC  FLAGS: NO ACCESSORS
FCB VECTOR: %0  %0
********************
(etc... for each file)
```

Scripts Corner (continued)

If you have a lot of files in your disc area, then you have a very cumbersome printout. Now lets say you need to manage the disc space. You only need to know the creator, date last accessed and the name of the file. Rather than search the bulky listing with the yellow pen, why not write a shell script to retrieve the data on one line per filename? That's what this one does:

script:

```
# creators --- filename, creator and access date listing

printf "listf @$1;pass" | listdir5.pub.sys >t1:temp

sedit -n -e '/FILE: /p' t1:temp >t2:temp
sedit -n -e '/CREATOR: /p' t1:temp >t3:temp
sedit -n -e '/ACCESSED: /p' t1:temp >t5:temp

filed 22-58 '$1' t3:temp >t4:temp
field 22-58 '$1' t5:temp >t6:temp

lam t3:temp t4:temp t6:temp
```

output:

```
FILE: LISA.TEST.AMDRD CREATOR: MGR ACCESSED: WED, 13 FEB 1985
FILE: OCLDIFA.TEST.AMDRD CREATOR: MGR ACCESSED: WED, 13 MAR 1985
FILE: OCLTA.TEST.AMDRD CREATOR: MGR ACCESSED: FRI, 15 MAR 1985
FILE: OCLTDIFA.TEST.AMDRD CREATOR: MGR ACCESSED: WED, 13 MAR 1985
FILE: OCLTRANA.TEST.AMDRD CREATOR: MGR ACCESSED: FRI, 15 MAR 1985
FILE: REFMANLA.TEST.AMDRD CREATOR: MGR ACCESSED: WED, 13 FEB 1985
FILE: SENDA.TEST.AMDRD CREATOR: MGR ACCESSED: TUE, 12 MAR 1985
FILE: SYNOPSIA.TEST.AMDRD CREATOR: MGR ACCESSED: TUE, 5 FEB 1985
```

The script places the appropriate command into the input stream for the program, which produces the normal output. This output is then stripped of all but the required lines by the next 3 commands. The garbage is stripped from each line by the field tool, leaving only the interesting data to laminated together on one line by lam. The results are as listed beneath the commands.

If this seems a trivial, round-about method of getting what you want, you may have a verygood point. Keep in mind, though, that this was put together in a matter of an hour or so. This included learning the tool functions that were available and their proper usage and syntax. How long would it have taken to write a program in our favorite languauge to perform the same function? Admittedly, the execution time is not the greatest, but then who's time is more valuable, your's or the machines?

Ideally, the machine is your tool. A script can help put the burden back on the beast and free for the creative challenges.

So, we have our first useful script, humble as it may be. Here's a challenge for you. Can you take this tools and extend it to delete those files with an access date greator than a particular value passed by a parameter?

I am interested in seeing your scripts put to paper. Write them down, regardless of which machine they are for or how sketchy they may be. Send them to me and I will put them in a future Scripts Corner for our readers enjouyment. Don't worry about the readability; I'll supply the witty words, you supply the techniques and together we'll make a contribution to the tools community.

Software Tools in Pascal on Micros

Willett Kempton
Princeton University

The source from the book *"Software Tools In Pascal"* by Kernighan and Plauger has been implemented under Turbo Pascal. The original implementation was done by Bill McGee on an Apple][with a CP/M-80 card. From his implementation, I adapted the primitives to run under CP/M-86 and MS-DOS, added pipes, sequential processes and (very limited) spawning of daughter processes. I also added a configuration section which makes it very easy to set up the system for any of the three operating systems under various hardware configurations (e.g. piping is done to a RAM disk if available, the default floppy if not).

Turbo Pascal produces efficient object code. The tools run noticably faster than they did under UCSD Pascal. The entire runtime system occupies 130 Kbytes on disk, compiled from 4500 lines of source. It has been tested under Turbo versions 1.0 and 2.0, on all operating systems, and has been run on a variety of hardware.

I have uploaded the source to CompuServe and to a couple of public-domain bulletin board systems. It has proved a popular upload; the CompuServe version alone has seen 250 downloads in just two months.

Time constraints prohibit me from answering individual inquiries or sending disks, so I suggest that interested tool users download from CompuServe (GO BAR at any ! prompt, then enter Borland SIG), or if a VT100, DEC Rainbow, or VT100 emulator is available, from Rainbow_Data: 213-204-2996). The file name is TPTOOL (Turbo Pascal Tools) or TPTOOL19.LBR. May 85

Translating Ratfor to C

Clyde Lightfoot

The following three articles were submitted by Clyde Lightfoot. The first is about his product, the *rtc* translator. The translator is in the final stages of development and testing and is scheduled for release by the end of 1985. If you have any questions or comments regarding the following materials, please contact:

Clyde Lightfoot
1683 Milroy Place
San Jose, CA 95124
(408) 448-3016

RTC - A *Ratfor* to *C* Translator

I. Why Translate *Ratfor* to *C* ?

When *Ratfor* was first developed as a preprocessor to FORTRAN, the C language, after which it was patterned, was not widely accepted and was available mainly on PDP's. FORTRAN, on the other hand was available on most machines and a relative standard subset could be found. *Ratfor* corrected many of FORTRAN's inadequacies, but added another layer to the compile and link process and put further distance between the programmer and the final product. In addition, FORTRAN was developed to crunch numbers, whereas *Ratfor* programs typically involved large amounts of character manipulation that were not well suited to FORTRAN. To correct this inherent inefficiency, assembly language additions to the *Ratfor* library became necessary, but in turn reduced portability.

In the ensuing 10 years, the C language has become widely accepted and is available on nearly every machine from micro to mainframe. Despite the fact that there is not yet an ANSI standard for *C* , it is considered by many to be more standard than FORTRAN across different machines and operating systems. It is often one of the first high level languages to be installed on a new machine or operating system. In contrast, FORTRAN is often one of the last languages to be offered, especially on micros. Market demand or the complexities in bringing up the compilers on a

new system may dictate this, but the result is that any programs dependent on FORTRAN will not be available for some time on a new system.

C is inherently very powerful. Programs are usually smaller both at the source code and binary levels and more efficent than *Ratfor*, because the programmer is one step closer to the machine. Once in *C*, programs can be optimized and future improvements can take advantage of *C*'s advanced constructs. With this in mind, and in view of the fact that *Ratfor* is modeled after *C*, it seems only natural to translate *Ratfor* based code to *C*.

Additionally, the Software Tools represent many man-years of effort with much useful code and algorithms. Many have studied the tools both for their own education and to enable them to duplicate the tools in other languages and on various systems. By offering such programs in *C*, more people will turn to them for instruction, and there will be a far greater reason to maintain and upgrade the tools. Translating these tools to *C* preserves the investment in time used to write and learne these programs.

A few of the tools have been hand translated into *C*, e.g., *format*, *macro*, and *ar*, but this is a very time consuming process that can lead to the introduction of errors. Some have concluded that it is easier to rewrite the tools from scratch rather than try to translate them from Ratfor. The problem is not due to differences between the *Ratfor* and *C*, but the inherent properties of the FORTRAN underlying the *Ratfor*. By machine translating *Ratfor* to *C*, it should be feasible for a small organization or individual to undertake the translation of some or all of the tools, without a great investment in time and money.

II. A Description of the *rtc* Translator

The *Ratfor* preprocessor logic was closely followed in the development of the *Ratfor to C* translator, in order to understand the full implications of the *Ratfor* statements. This was especially true in the somewhat grey area of line continuation. In other areas, the error checking was increased based on what was expected in a *Ratfor* statement. This was done because the philosophy of letting the FORTRAN compiler catch the error did not always apply where the *C* compiler was the target environment.

There is an emphasis on developing an interactive translating machine that detects errors or translation differences during the translation process and gives the user a chance to correct them immediately, in contrast to generating a list of errors or hand changes that are later required. Although most of the translation is expected to be done with the assistance of the machine, there is some *Ratfor* and FORTRAN code which is not supported and is simply passed through. There is also a means for the user to designate that a line of code be passed though the translator when it is encountered as an error, rather than forcing the user to fix it there on the spot.

There are three steps involved in this implementation of a *Ratfor to C* translator. The first is a preprocessing step that performs a limited degree of translation by the use of translation macros and selected translation of certain *Ratfor* items, such as escaped characters and operators. Although limited, a large degree of customization can be performed in the preprocessing step, such as renaming function calls to *C* equivalents and inserting prefixes specifying certain arguments to be passed by value instead of by address, the FORTRAN default. Additionally, certain macro replacements necessary for a proper translation, particularly those macros containing non-macro alphas that thus must represent variables or functions, are made here. The preprocessor automatically processes all include files to the same degree and builds a list of the include files followed by the *host* file being translated for use by the translator. The include files are not included in the *host* file though, but retain their identity to maintain the flexibility of the original source. Additionally, comments and spacing in the original source are preserved as well.

The next step is referred to as *pass1*. It performs the identification of all the variables, arrays, functions, and subroutines. This includes determining the dimensioning and initialization information for the variables and arrays. Although *Ratfor* code is supposed to

declare all variables and functions, because the underlying FORTRAN does not require this, certain variables and functions do slip through from time to time. This step will catch the undeclared items and promote character functions to integer and real functions to double. In order to properly indentify all functions and variables, *pass1* also keeps track of all macros that were not replaced in the preprocessing phase in order to distinguish them from 'real' code. The included files are processed first in the order they are encountered in the host files, and their declarations and macro definitions saved according to the file in which they were declared, so they can be activated whenever the include file is 'included'.

Additionally, *pass1* performs a large degree of error checking to inform the user of problems before much processing time has elapsed. (If an error is detected, the user may correct it and then *pass1* will start over with the routine where the error was detected, rather than the entire file.) Upon detecting an error, *pass1* will enter a simple full screen editor that allows the user to make the correction (but only in the function or subroutine where the error occurred), or abort the translation. If a change is made and the program is instructed to continue the translation, *pass1* will then re-analyze the current routine and if no further errors are detected it will then initiate the second pass for the current routine.

The final step is called *pass2* . This is where all the translation of the *Ratfor* to *C* will take place. It inserts declarations based on the variable identification of *pass1* , then deletes all of the FORTRAN (and *Ratfor*) declarations and initializations. Translations are made at the control statement level as well as at the individual item level. The translated file is output from *pass2* as it is translated. Errors detected in *pass2* cannot be corrected and unfortunately result in immediate termination of the translator.

In effect there is one more step in the translation process that involves the definition of *C* compile time macros. There are some elements of *Ratfor* or FORTRAN that the author liked and presumed would be helpful toother users. Rather than translate them to their true *C* counterparts, they are modified as necessary and left in the code being translated. In all cases, these items are expressed as macros that must be replaced with the correct translation when the translated code is compiled in *C* . There aren't many cases of them, but the following are two examples. The **repeat-until** statement is left in tact and is defined as a **do-while** by *C* macros. The keyword **FUNCTION** is used in the declaration of a function (or subroutine) in order to make it easier for the user to see where new functions begin in *C*. This must be replaced with nothing at *C* compile time since it has no meaning in *C* .

III. Use of the *rtc* Translator

In general, it is always best to translate running *Ratfor* code. Make sure the program is running properly in its *Ratfor* form before trying to run it through the translator. Although the translator implements many error checks, confusion might arise if undebugged code is sent though. After the code has gone through the translator, hand fixes will be required for any literals that were passed through as well as for any unsupported statements. These include the *read* , *write* , and *format* statements, *equivalence* , *internal functions* , and any *complex numbers* , along with any non-standard FORTRAN-IV statements. After the translated code is compiled in *C* , a lint utility should be used to check for any other errors that slipped though the translation process. If the user then wants to optimize the new code, it would make sense to first run it through a profiler to determine where the optimization would be most beneficial.

To improve the translation results, the user should also review the preprocessor's translation macros and either add to or change certain definitions to accomodate different *Ratfor* or *C* environments. (Most of the Software Tools macros describing the operating system environement should be left intact through the translation process.)

IV. Summary

The *Ratfor to C* translator is a unique new product which will be especially useful for those in the software development field. It will provide greater flexibility in the use of current *Ratfor* code and will offer very appreciable time savings in the translation process. In particular, it will greatly benefit those seeking to translate the Software Tools for commercial, educational and personal use.

Expressing *Ratfor* in *C*

I. Introduction

The philosophy chosen for the translation of *Ratfor* into *C* is a strict, literal translation with minimal optimization done during the translation process. This is better suited to automation and requires minimal human intervention. Any optimization that is desired can be done to the translated code in *C* and applied to areas that will be most beneficial. By taking this approach, it is more likely that some version of the translated code will be up and running sooner. It should also be pointed out that the translation process does not change the code's functionality. For example, if the *Ratfor* preprocessor is translated into *C* , it will still process *Ratfor* into FORTRAN, not *C* !

The translations performed on the *Ratfor* and FORTRAN statements by the *rtc* translator are presented below. The *Ratfor* statements are obvious to most, and are the easiest to translate. On the other hand, the FORTRAN statements provide certain challenges for the translator and serve to reinforce the need for machine assistance in the translation process due to their subtleties.

II. Translation of the Ratfor Statements

The *Ratfor* statements basically mimic their *C* counterparts with a few exceptions, with the most basic difference being the method of continuing lines. A line in *C* is automatically continued across lines until being terminated by a semicolon, whereas *Ratfor* uses implied continuation. Generally speaking, if a *Ratfor* statement ends in a comma, open parentheses, or operator, it is assumed to be continued across to the next line. However, in certain *Ratfor* statements, logic that is used to balance parentheses transcends end of lines. Additionally, a semicolon can be used to mark the end of statements and an underscore immediately followed by a newline indicates a line continuation. This same logic must be used in the translation process in order to determine the end of a statement and when to place the semicolon required for *C* . Given that the *Ratfor* continuation logic is followed, the translation of *Ratfor* control statements is presented in TABLE 1.

Most of the *Ratfor* statements do not present special problems, but some deserve special mention. The **repeat-until** statment is readily translated into a *C* **do-while**, but unlike *Ratfor* , the **while** s not optional in *C* and must be added with a true expression to duplicate the forever function of the **repeat** statement. The *Ratfor* (and FORTRAN) **do** statements lend themselves to translation to the **for** statement, but this is not a literal translation since the *Ratfor* /FORTRAN-IV **do** statement will always execute once even if the test fails on the first loop. The **for** loop tests at the top and will not execute if the test fails. Although a true literal translation of the **do-loop** can probably be devised with a combination of if, **goto**, and increment statements, this property of the **do-loop** often causes problems and must be programmed around with an additional external if test, hence I have opted for the **for** statement as the **do-loop** translation. The *Ratfor* **switch** statement translates in the *C* form with the addition of **break** statements and the splitting up of the *Ratfor* **case** constants into multiple **case** statements as shown in the table. The normal *Ratfor* **break** statement is identical in function to its *C* counterpart, however the optional number following the **break** statement is not supported and must be hand translated. (The **break n** form is not used very often and should not pose a problem.) Similarly, the *Ratfor* **next** statement is identical to the *C* **continue** statement. Finally, any *Ratfor* literal statement or fragment is simply passed through the translator untouched, since it presumably contains non-standard FORTRAN-IV code.

TABLE 1
Ratfor Control Structures

Ratfor	*C*
`if( condition )` `  statement(s)`	`if( condition )` `  statement(s);`
`else` `  statement(s)`	`else` `  statement(s);`
`while( condition )` `  statement(s)`	`while( condition )` `  statement(s);`
`for(init; condition; incr)` `  statement(s)`	`for(init; condition; incr)` `  statement(s);`

`repeat`
`  statement(s)`
`until( condition )`
	`do`
	`  statement(s);`
	`while( !(condition) );`

`do a=b, c, d` `  statement(s)`	`for(a=b; a<c; a=c+d)` `  statement(s);`

`switch( expression )`
`  {`
`  case const1, const2:`
`  statement(s)`
`  case consta, constb:`
`  statement(s)`
`  .`
`  .`
`  .`
`  default:`
`  statement(s)`
`  }`
	`switch( expression )`
	`  {`
	`  case const1:`
	`  case const2:`
	`  statement(s);`
	`  break;`
	`  case consta:`
	`  case constb:`
	`  statement(s);`
	`  break;`
	`  .`
	`  .`
	`  .`
	`  default:`
	`  statement(s);`
	`  break;`
	`  }`

`break`	`break;`
`break n`	`(not supported, repl w/ goto label`
`next`	`continue;`
`{}`	`{}`
`[]`	`{}`
`% literal_statement`	`LITERAL literal_statement`

`fragment1 %( fragment2 %)`
`  fragment3`
	`fragment1`
	`BEG_LITERAL fragment2 END_LITERAL`
	`  fragment3`

`string name "text"`	`char name[] = "text";`
`string name(n) "text"`	`char name[n] = "text";`

The macros **literal**, **beg_literal**, and **end_literal** replace the **%** and **percent-parenthesis** combinations in the translated code. These strings may be simply defined as nulls at *C* compile time if the code is ok for *C*, or left undefined for the *C* compiler to identify as undefined variables. The literal features also provide a convenient means of delaying the correction of errors found during the translation until later.

TABLE 2 gives the translation of the FORTRAN-IV and *Ratfor* types into *C* , and there are three things to mention here. First, even though there isn't a logical type in *C* , integers should serve this function well. However, in case the logical variable is being used to reserve some space, the logical variables are translated into macros or typedef names that preserve the original size information about the logical. Then at *C* compile time, these macros can be defined to be the equivalently sized integer or they can be defined as **short** s, thus preserving space. Secondly, the *Ratfor* double integer macros are replaced with equivalent **long** expressions. Finally, complex numbers are not supported. Even though it is easy enough to define a structure to hold a complex number, all of the operations involving complex numbers must be replaced with function calls, which would add an inordinate amount of complexity to this product for a minimal return.

The *Ratfor* operators are the same as *C* with the exception that ^= and ~= are also accepted for !=, and | and & must be changed to || and && respectively. Additionally, the FORTRAN exponential operator must be translated into a call to the *C* **pow** function. The FORTRAN logical and relational operators are also translated to their *C* counterparts.

Translation of the *Ratfor* character escape sequences is given in TABLE 3. In most cases, all that is involved is simply changing the **at-sign** into a **backslash**. Note, that the **double at-sign** reduces to one, and the **backslash** become a **double backslash** .

TABLE 4 gives the translations performed for the *Ratfor* preprocessing commands. There are several items worthy of note here. The case of a macro definition with arguments

TABLE 2
Type Translations

FORTRAN-IV	*C*
LOGICAL	LOGIC -> macro for 'short'
LOGICAL*1	LOGIC -> macro for 'short'
LOGICAL*2	LOGIC2 -> macro for 'int' or 'short'
LOGICAL*4	LOGIC4 -> macro for 'long' or 'short'
INTEGER	int
INTEGER*1	short
INTEGER*2	int
INTEGER*4	long
REAL	float
REAL*4	float
REAL*8	double
DOUBLE PRECISION	double
COMPLEX	(not supported)
COMPLEX*16	(not supported)
R4,	

CHARACTER	char
(double integer macros)	long

TABLE 3
Character Escape Sequences

Ratfor	*C*
'@n'	'\n'
'@t'	'\t'
'@@'	'@'
'@r'	'\r'
'@b'	'\b'
'@e'	'\0'
'@f'	'\f'
'@l'	'\n'
'@ddd'	'\ddd' (octal digits)
'\'	'\\'
'@"	'\"

Note: all other *Ratfor* character literals would be the same as *C* .

TABLE 4
Preprocessing Commands

Ratfor	*C*
include filename	#include "filename"
include "filename"	#include"filename"
define(name,repl_str)	#define name repl_str (w/o arguments)
define (name,repl_str_w/args)	#define name(z_1,...,z_9) repl_str_w/z_1,etc)
undefine(name)	#undef name
ifdef(name)	#ifdef name
elsedef	#else
enddef	#endif
ifnotdef(name)	#ifndef name
arith(x,op,y)	(x)op(y)
incr(x)	(not supported)
ifelse(a,b,c,d)	"
substr(s,m,n)	"
lentok(str)	"

requires that argument names be created and used in place of the dollar sign number combinations in the replacement text and be added to the macro name in an argument list. It was decided to merely replace the **$** with the string **z_** in order to create the macro argument name. There are no checks to ensure that the created names do not already exist in the replacement text, since it was felt that the number of times a **z_1** or **z_2** would show up as an individual item in the replacement text was minimal at best. (If you suspect these combinations may show up in some macros, it is best to change them before the translation begins.) Additionally, any macro which is replaced with an alpha(that in turn is not a macro), will be replaced during the preprocessing phase of the translation in order to make intelligent translations in *pass1* and *pass2* . The last five *Ratfor* preprocessing commands have no equivalent in *C*. However, since *C* evaluates constant expressions at compile time, the arith command can simply be replaced with the equivalent arithmetic expression during the translation process. The other commands, **incr**, **ifelse**, **substr**, and **lentok**, are currently not supported, (but I am considering replacing them in the preprocessing phase of the translation with their currently defined conditions). Finally, the *Ratfor* conditional preprocessing commands, (e.g., **ifdef**, **enddef**, etc.) will normally be removed during the preprocessing phase of the translation along with their corresponding code according to the current definition of macros present in the code. This is necessary in order to do proper error checking of parentheses and braces in pass1 and pass2 . It is intended to include an option to deactivate error checking and the conditional preprocessing not be evaluated in order that these preprocessing commands and corresponding code be passed though the preprocessor and translated by *pass1* and *pass2* , (however, unpredictible results can be expected).

III. Accounting for the Characteristics of FORTRAN-IV

The differences in the characteristics of the FORTRAN-IV underlying the *Ratfor* and *C* require many subtle translations and can prove to be very challenging. They include such things as static versus automatic variables, passing arguments by address rather than by value, the order and range of array indices, implicit typing versus total declaration in *C* , commons versus externs, equivalence versus pointers or unions, real precision arithmetic versus double precision arithmetic in *C* , case dependence in *C* , etc., etc. Most of the these can be compensated for to one degree or another. Some things like *C'* s use of only double precision arithmetic in expressions cannot be changed without some reprogramming of expressions to reduce their accuracy. Luckily, the increased accuracy offered by *C* is usually not a problem, and in fact is often desired.

Often there is a similar, but not exact representation in *C* of a FORTRAN statement, such as the case of FORTRAN's common and *C* 's extern declaration. In most cases, the extern is sufficent to represent the features of the common statement, mainly the transfer of information between routines other than by arguments. However, when a common variable is given different names in different invocations of the common statement among

routines, the extern analogy breaks down. Similarly, when a local array is equivalenced toa location of a scalar in a common and the programmer is counting on accessing other elements in the common, the analogy also breaks down. It is possible to come up with an exact translation of the common statement through the use of pointers and assigning those pointers to different offsets in a block of memory equal in size to the original common, but this is a fairly complex operation and it was decided that the extern analogy would be better instead, (especially since the *Ratfor* code I examined involved a clean use of commons suitable to the extern analogy).

The case of equivalences is different however. The obvious *C* analogy is a union statement, however it turns out to be a subset of the equivalence. A better analogy is a pointer to one or more of the equivalenced variables. Unfortunately, the logic required to generically duplicate the expected equivalence uses is very complex, (you basically have to duplicate what the FORTRAN compiler must do), and since the equivalence is not used that often in the *Ratfor* code I have seen, it was decided to not support the equivalence statement at this time.

Fortunately, most of the other differences have rather straight forward solutions. For example, all local variables except arguments are declared as statics. Any implicitly typed variables are declared according to the typing rules in effect at the time of the translation, and all variable names are converted to lowercase.

Translations for the FORTRAN-IV statements normally used in *Ratfor* programs are shown in TABLE 5. Any FORTRAN statments not recognized by the translator will be considered 'other' and treated with logic similar to the *Ratfor* preprocessor's 'otherc'/'eatup' combination.

IV. Summary

The *Ratfor* to *C* translations presented here cover most of the translations performed by the rtc translator, excepting some details omitted for brevity. They are suitable for use when translating by either hand or machine, but lend themselves to machine translations for which they were developed. Following them rigorously should result in a fairly good translation, but could prove to be very tedious work if you are translating by hand. Of course you can always break up the boredom by throwing in some custom optimizations - replacing *linepointer s* with true pointers perhaps!

TABLE 5
FORTRAN-IV Statements

FORTRAN-IV	C
type function name(arg_list)	c_type FUNCTION name(arg_list) arg_list declarations; { local declarations;
subroutine name(arg_list)	VOID FUNCTION name(arg_list) arg_list declarations; { local declarations;
common/label/var_list /label2/var_list	extern /*label*/ ctype var_list, /*label2*/var_list; Note: plus the externs must be declared outside of a routine once.
call subr(arg_list)	subr(arg_list); (note: '()' are mandatory)

do n a=b,c,d
statement(s)
n statement
	for(a=b; a<c; a=c+d){
	statement(s);
	ln: statement;
	}

goto n	goto ln;
n statement	ln: statement; ('n' is a number and 'l' is the letter 'L')

continue
stop
end
	;
	exit();
	}

read write format equivalence	(not supported) " " "

Note: all FORTRAN-IV type, dimension, & data statements are read for their typing information, but are deleted and replaced with the C equivalents in the function or subroutine declaration, which combine type, dimension, & data statement information.

Proposed Changes to the Software Tools library in *C*

There are close to 100 tools in the Software Tools system. To attempt to optimize all of them in *C* would be a big job. However all of the tools will be enhanced by optimizing the library and primatives in *C* .

This optimization should take the form of improving the internals of library routines while maintaining the same calling convention of the routines. Areas that will benefit most will be those involving strings where pointers can be used instead of arrays and indexes. Additionally, the area of of memory allocation should be made both more efficient and flexible with *C* 's capabilities.

If it is desired to change the calling convention, then a new function name should be created and used in future improvements to the tools themselves. In many cases, this may simply mean using a standard *C* library function rather than inventing another.

There is one change to the calling convention that the *rtc* ranslator will support, namely pass by value. A prefix can be added to an argument, (via the use of a translation macro), that will inform the translator to pass the value of the argument rather than its address, as is the regular FORTRAN convention. If this is done via a translation macro and that macro is used in the translation of all of your *Ratfor* code - both tools and library - then a consistent translation will be performed. However, in order for this not to change the logic of the code, it can be applied only to those library functions that do not change the value of the argument. Routines that do and don't meet this criterion are stated in the library manual.

A committee or forum should be established to identify which library routines should be optimized first, along with specifing new routines in *C* . This isn't a very difficult job, however the persons involved in the effort should have a good working knowledge of how the tools are written.

INTELLIGENT DECISIONS, INC.

P.O. Box 50174
Palo Alto, California 94303
408/996-2399 Telex 658340 Intertel SNC Cable Contel

OTHER PRODUCTS FROM INTELLIGENT DECISIONS

Software Tools Catalogue
Produced in cooperation with the Software Tools Users Group, this catalogue is a treasure chest of information about translators, Ratfor, macro preprocessors, new tools, shells, text formatters, translations and more. The tools run on over 35 different machines. The catalogue includes a list of people to contact regarding implementations on specific machines, putting you in touch with people on the "front lines" and enabling you to avoid duplication of effort.

UUCP Network Directory
This directory lists over 1,200 people who are reachable via this rapidly growing international network of over 3,500 machines and over 25,000 people. Each entry lists the person's name and network address and in many cases their work phone and something about them.

MINIMACS
A small model Gosling EMACS for UNIX which includes many of the features of EMACS but does not include the built-in MLISP programming language. MINIMACS is available on all implementations of UNIX for a wide range of computer systems.

ORDER FORM

____ EMACS Machine Type ____		OS _____	
☐ UNIX	☐ Binary $395	☐ Source $995	
☐ VMS	☐ Binary $2,500	☐ Source $7,000	
☐ MS/DOS	☐ Binary $325	☐ Source $995	$_____
____ MINIMACS Machine Type ____		OS _____	
☐ VAX	☐ Binary (N/A)	☐ Source $795	
☐ MC68000	☐ Binary $395	☐ Source $795	$_____
____ SOFTWARE TOOLS CATALOGUE(S)		$8.95 each	$_____
____ UUCP NETWORK DIRECTORY(IES)		$8.95 each	$_____
____ 4 ISSUES UUCP NETWORK DIRECTORY		$29.95/yr	$_____
		Subtotal	$_____
Sales tax (Calif. residents add applicable tax)			$_____
Shipping & handling: Domestic $2.00/Overseas $5.00			$_____
		Total	$_____

Payment may be made by check, money order, VISA or MasterCard.

Credit card #_____ Exp. date_____

PO # (For orders over $250 only)_____

Signature_____

Name_____

Address_____

City_____ State_____ Zip_____

Send orders to: Intelligent Decisions, Inc., PO Box 50174, Palo Alto, CA 94303 0174, USA

_____Please send me information on other products from Intelligent Decisions.

software tools users group

140 center street, el segundo, ca 90245

Software Problem Report

Date:_____

Originator's Name:_____ Address:_____

Phone:_____ _____

Net Address:_____ _____

_____ _____

Name of Tool(s):_____

Machine/Operating System:_____

Date of Distribution Tape:_____

****

Problem Description: (Check all that apply) ____source ____routines ____manual entry

****

****

Suggested fix, if any:

****

If you have questions, call the
STUG Hotline: (415) 486-4680

****

software tools users group

140 center street, el segundo, ca 90245

Software Submission Form

Please read the article on Tape Submission included in this newsletter. Your submission should be in archive format and include manual entries, routines, etc. as described in the article.

Machine and System on which you made the tape:

Brief description of tape contents:

Density: _____800 bpi _____1600 bp
(9-track only)

Character Code: _____ EBCDIC _____ ASCII

Blocking Factor:

Software Release
****

I (We) the undersigned give the Software Tools Users Group permission to reproduce and distribute all or any part of the program package material contained on the above tape for the use of STUG members. This material is not subject to copyright.

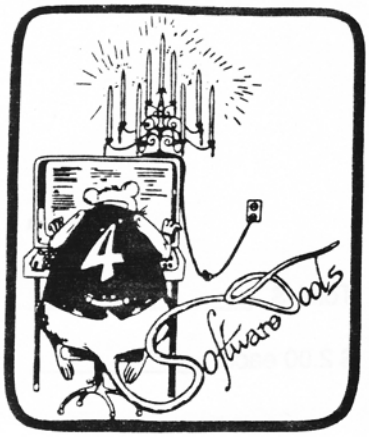
Submitted by: _____

_____

_____

SIGNATURE(s) _____ DATE _____

_____ _____



software tools users group

140 center street, el segundo, ca 90245

Order Form

General Information

Date: _____

Name: _____

Address: _____

_____

City: _____ State / Zip: _____

Country: _____

Phone: _____ Network: _____

Privacy: _____ Yes, I wish my name, address, phone, etc. to be kept private.

_____ No, I do not request that this information be kept private.

System Information

Machines and systems on which you use the Software Tools:

_____

Utilities/library functions which you have implemented:

_____ The standard package (as distributed by STUG)

_____ The original package (Kernighan and Plauger)

_____ Other: _____

Other systems on which you plan to implement the Tools:

_____

Special Interests:

_____

** Please Turn Over **

Membership

Renewal

____ Individual Membership $ 20.00 /yr
____ Corporate Membership $ 200.00 /yr (includes 1 tape)
____ Sustaining Membership $ 2000.00 /yr

• Number of years _____ Foreign Delivery $ 5.00/yr Total $ _____

_____ Single Newsletters (#1 Nov. 79 - #13 No. 84) $ 2.00 each _____

Tape Orders

*** Membership is now required**

Target computer(s) for the tools: _____

Tapes Available	Specify Density		Quantity
Basic Tapes			
82 Portable LF Terminated, 2048 cpb ASCII	800	1600	_____
82 Portable Card Image, 3200 cpb ASCII	800	1600	_____
84 Toys Tape (Card Image, LF Term, or VMS format)	800	1600	_____
Specific Implementations			
84 VAX/VMS (Backup Format)	800	1600	_____
83 RSX-11M (BRU Format)	800	1600	_____
82 TOPS-20	800	1600	_____
82 Unix 4.1 BSD (Tar Format)	800	1600	_____
83 IBM/CMS	800	1600	_____
83 IBM/MVS	800	1600	_____
83 UNIVAC 1100	800	1600	_____
? SEL MPX	800	1600	_____
83 HP1000 RTE-IVB + 6/VM (READR or fc format)	800	1600	_____
84 Carousel Tools for CP/M *			_____
85 Carousel Tools for MS-DOS **			_____

• Number of tapes _____ Foreign Delivery $ 10.00/yr Total $ _____
@ $80.00 each

* Requires Microsoft FORTRAN to use Ratfor, or compile the public domain tools
** Requires IBM FORTRAN to use Ratfor, or compile the public domain tools

software tools users group

140 center street, el segundo, ca 90245

STUG Asks You

Questionaire

What computer conferences do you attend?

USENIX DECUS UNIFORUM
Others: _____

What systems do you use which have the tools?

VMS RSX RSTS TOPS-10/20
Unix™ IBM/CMS IBM/MVS HP/RTE
MS-DOS CP/M Univac SEL MPX
Other: _____
_____

Whose tools do you use?

STUG U. of Arizona Ken Poulton
Georgia Univ. Prime Users Grp. Carousel
Your own
Others: _____
_____

Do you consider yourself a:

User — One who **only uses** the tools
OR Implementor — One who uses **and** modifies or writes new tools
&/OR System Installer — One who **installs** the tools

Have you developed any new tools?

Yes No
Describe: _____
Language: _____

What would you be able to help STUG with? _____
_____

What would you like STUG to do or provide you with? _____
_____