NAME

RATFIV PRIMER

Ratfiv is a preprocessor for Fortran. Its primary purpose is to encourage readable and well-structured code while taking advantage of the universality, portability, and efficiency of Fortran. With modest effort Fortran-based programmers can increase their productivity by using a language that provides them with the control structures and cosmetic features essential for structured programming design. Debugging and subsequent revision times are much faster than the equivalent efforts in Fortran, mainly because the code can be easily read. Thus it becomes easier to write code that is readable, reliable, and even esthetically pleasing, as well as being portable to other environments.

Ratfiv allows for all the features of normal Fortran, plus makes available these control structures:

"if"-"else" conditionals
"while", "for", "do", and "repeat"-"until" looping constructs
"switch" case statement
"break" and "next" for exiting loops
statement grouping with braces

The cosmetic aspects of Ratfiv have been designed to make it concise, eay to maintain, and pleasing to the eye:

"include" statement for including source files
"return value" statement in functions
formats specified within read, write, encode, and decode statements
"string" statement for initialization of character arrays
specification of numbers in bases 2-36
free form input
unobtrusive comment convention
translation of >, <=, etc. into .GT., .LE., etc.
quoted character strings
conditional compilation

In addition, Ratfiv provides a sophisticated macro processor for the definition of symbolic constants as well as macros with arguments.

Ratfiv is implemented as a preprocessor which translates the above features into Fortran (optionally VAX Fortran 77), which can then be fed into almost any Fortran compiler. Ratfiv programs may be written using upper or lower case, however Ratfiv keywords (such as "if", "else", etc.) must appear entirely in upper or lower case, e.g. "If" is not recognized as an "if" statement. The section of this document titled "Using Ratfiv" tells how to invoke Ratfiv on your system.

Each of the Ratfiv features will now be discussed in more detail. In the following, a "statement" is any legal statement in Fortran: assignment, declaration, subroutine call, I/O, etc., or any of the Ratfiv statements themselves. Any Fortran or Ratfiv statement or group of these can be enclosed in braces-- { } --or brackets-- [ ] --to make it a compound statement, which is then equivalent to a single statement and usable anywhere a single statement can be used.

IF-ELSE

Ratfiv provides the "if" and "else" statements to handle the construction "if a condition is true, do this thing, otherwise do that thing". The syntax is

```
if (legal Fortran condition)
        statement(s)
else
        statement(s)
```

where the else part is optional. The "legal Fortran condition" is anything that can legally go into a Fortran logical IF. The Ratfiv statements may be one or more valid Ratfiv or Fortran statements of any kind. If more than one statement is desired, the statements must be enclosed by braces. For example,

```
if (a > b)
{
  k = 1
  call remark (...)
}
else if (a < b)
{
  k = 2
  call remark (...)
}
else
return
```

WHILE

Ratfiv provides a while statement, which is simply a loop: "while some condition is true, repeat this group of statements". The syntax is

```
while (legal Fortran condition)
        statement(s)
```

As with the if, "legal Fortran condition" is something that can go into a Fortran logical IF. The condition is tested before execution of any of the Ratfiv statements, so if the condition is not met, the loop will be executed zero times. Also, as with the IF, the Ratfiv statements can be any valid Ratfiv or Fortran constructs. If more than one statement is desired, the statements must be enclosed by braces. For example,

```
WHILE (GETC(C) != EOF) [
        C = CNVT(C)
        CALL PUTC(C)
        ]
```

Note that upper case is perfectly acceptable to Ratfiv, and that brackets ([]) may be used instead of braces ({}).

FOR

The "for" statement is similar to the "while" except that it allows explicit initialization and increment steps as part of the statement. The syntax is

```
for (init clause; condition; increment clause)
        statement(s)
```

where "clause" means a single Fortran statement or else a group of Fortran statements enclosed in brackets and separated by semi-colons. The "init clause" is executed once before the loop begins. The "increment clause" is executed at the end of each pass through the loop, before the test. "Condition" is again anything that is legal in a logical IF. As with the "while" statement, the condition is tested before execution of any of the Ratfiv statements, so if the condition is not met, the loop will be executed zero times. Any of init, condition, and increment may be omitted, although the semicolons must remain. A non-existent condition is treated as always true, so "for(; ;)" is an indefinite repeat. The "for" statement is particularly useful for chaining along lists, loops that might be done zero times, and similar things which are hard to express with a DO statement. Here are two examples of "for" loops:

```
for ({ nch = 0; i = 1 }; getarg(i, file, MAXLINE) != EOF; i = i+1)
{
  int = open (file, READ)
  while (getlin (line, int) != EOF)
  for (j = length(line); j > 0; { j = j-1; nch = nch+1 })
    call putc (line(j))
  call close (int)
}
```

The above code reads from a list of files and writes each line from each file out backwards. The variable "i" keeps track of the number of files read and the variable "nch" keeps track of the number of characters written out. The "!=" means .NE.

DO

The "do" statement is like the Fortran do-loop. The syntax is:

```
do  Fortran do-part
      statement(s)
```

The Fortran do-part is whatever Fortran will accept after a do, with the exception of the statement label. If more than one statement is desired, they must be enclosed in brackets. For example:

```
do i = 1, 4
{
  do j = 1, 4
  x(i,j) = 0.0
  x(i,i) = 1.0
}
```

This example initializes a 4 by 4 matrix to the identity matrix.

REPEAT-UNTIL

The "repeat" and "until" statements allow for repetition of a group of statements until a specified condition is met. The syntax is:

```
repeat
        statement(s)
until (condition)
```

The "until (condition)" is optional. "Condition" is again anything that is legal in a logical IF. The "condition" is tested at the end of the loop, so that at least one pass through the loop will be made. If the "until (condition)" part is omitted, the result is an infinite loop which must be broken with a "break" or "next" statement (see below). Once again, if more than one statement is desired, the statements must be enclosed by brackets. An example of a repeat-until loop is:

```
repeat
{
  call putc (BLANK)
  col = col + 1
}
until (col >= tabpos)
```

The example always puts at least one BLANK.

SWITCH

The switch statement is a multi-way decision maker that allows selection of one path of execution from among many based on the value of an expression. The syntax is:

```
switch (legal fortran expression)
{
  case expression(s):
    statement(s)
    .....
  default:
    statement(s)
}
```

The legal fortran expression in parentheses is evaluated and then tested to determine which of the constant expression(s) in each case statement it matches; when a matching value is found, execution is started at that case. Once the statements at the matching case have been executed, execution continues after the "switch" statement. The case labelled "default" is executed if none of the other cases match. The "default" is optional; if omitted and none of the cases match, no action is taken and execution falls through to the end of the case statement. "Expression(s)" may be a list of constants separated by commas, a single constant, or a range of constant values separated by a "-" (minus) character. Constants must be integers, character constants of the form ’c’ or "c", or an escaped character constant. (See the section on the "string" statement for a description of escaped character constants.) In addition, """" stands for a literal double quote, and ’’’’ stands for a literal single quote. Cases and defaults may occur in any order, and must be different. Note that enclosing brackets are necessary after the "switch (legal Fortran expression)" and at the end of the "switch" statement. Brackets are NOT necessary after a "case" or "default". Following is an example of the "switch" statement:

```
switch (lin(i))
{
  case 0:       return          # end of line
  case "0"-"9":
    j = ctoi(lin, i)    # convert to number
    k = max(j, k)
  case "A"-"Z", "a"-"z":
    ....                    #   do something
    ....                    #
  default:
    write(5, (’ error at lin(’,i3,’)’)) i
}
```

Ratfiv is able to determine whether a given "switch" statement is best implemented as a Fortran computed goto or as successive logical if tests. There is one possible problem with the "switch" statement; it generates a variable beginning with the letter "i". If variables beginning with "i" are implicitly declared other than integer (by an "implicit real (a - z)" statement, for instance) then the "switch" statement will not work properly in some cases.

BREAK and NEXT

Ratfiv provides statements for leaving a loop early and for beginning the next iteration.

"Break" causes an immediate exit from whatever loop (which may be a "while", "for", "do", or "repeat") it is contained in . Control resumes with the next statement after the loop. If "break" is followed by a number, then that many enclosing loops are exited, otherwise the loop in which the "break" appears is exited. For example:

```
repeat
{
  if (getc(c) == EOF)  break
    ....
}
```

"Next" is a branch to the bottom of a loop, so it causes the next iteration to be done. "Next" goes to the condition part of a "while" or "until", to the top of an infinite "repeat" loop, to the increment part of a "for", and to the next iteration of a "do". If a number is specified after the "next", then control is given to the loop which is that many nested loops out from the "next", otherwise control is given to the enclosing loop. For example:

```
for (i = 1; i < 10; i = i+1)
{
  if (array(i) == BLANK)  next
    ....
}
```

The "next" causes a branch to the increment part of the "for" loop, which adds 1 to "i".

RETURN

The "return" statement may be used as usual. However, in a function subprogram the function value may be implicitly assigned before returning using the following syntax:

```
return value
```

Here "value" is the value of the function subprogram. For example:

```
integer function index(char,strng)
byte strng(80), char

for (i = 1; i <= 80; i = i+1)
      if (strng(i) == char)
            return i
return 0
end
```

FORMAT STATEMENTS

Format specifications may be included in read, write, encode, and decode statements by including the specification, surrounded by parentheses, in the same position in the statement as the format statement number would normally appear. For example:

```
write(5, ('$FILE? '))
read(5, (80a1), end=99) file
```

Formats may be specified in the usual way by specifying a normal statement number instead of a parenthesized format specification. For example:

```
read(5, 10, end=99) file
    10 format(80a1)
```

STRING

The "string" statement defines the size and contents of a character array. The syntax is:

```
string name "character string"
```

or

```
string name(size) "character string"
```

The first form of the string statement defines "name" to be a character (default byte on DEC systems) array big enough to accommodate all the characters in the character string plus a terminating end of string character (default 0 (nul) on DEC systems).

The second form allocates "size" characters for "name" and initializes it to the specified character string with the EOS character. "size" must be large enough to hold the entire string plus the terminator, and may be larger.

Single quotes may be used instead of double quotes to delimit the character string. As with quoted literals, two adjacent double quotes occurring within a double-quoted string are interpreted as a literal double quote, and two adjacent single quotes within a single quoted string are interpreted as a literal single quote.

The character "@" in a string has a special meaning. It and the character following it are replaced by one of the special characters shown below depending on the character following the "@":

```
@@       is replaced by an @
@B or @b is replaced by a backspace
@E or @e is replaced by an end-of-string (default 0 on DEC systems)
@F or @f is replaced by a form feed
@G or @g is replaced by a bell
@L or @l is replaced by a line feed
@N or @n is replaced by a NEWLINE (default line feed on DEC systems)
@R or @r is replaced by a carriage return
@T or @t is replaced by a tab
@V or @v is replaced by a vertical tab
@$       is replaced by an escape
@digits  is replaced by the octal value of "digits".
```

An "@" followed by any character not listed above is replaced by the character following the "@". The exception is when "@" is before the terminating quote character; in this case the "@" is interpreted literally as an "@", and has no special meaning.

All string statements must appear together after all normal Fortran declarations and before any DATA statements. More than one string may be declared in a string statement if the declarations are separated by commas.

Examples:

```
string error '@gError reading file. Execution terminated.'
string infile(FILENAMESIZE) "TI:", outfil(FILENAMESIZE) "TO:"
string del "@177"   # octal 177 is the ASCII "delete" character
```

Notes for advanced users:

The default values used to replace "@e" and "@n" in strings (see above) are 0 and 10 on DEC systems. These defaults may be overridden by defining the symbols EOS and NEWLINE, respectively, to be either an integer or a quoted character which will then be used instead of the default value.

Similarly, the default data type for strings (byte on DEC systems) and the default terminating character (0 on DEC systems) may be changed by redefining the symbols EOS and character, respectively.

Example:

```
define(character,integer)
define(EOS,-1)
        ....
string comnds "READ@eWRITE"
```

defines comnds as an integer array of length 11 and produces data statements which initialize it. Comnds(5) is initialized to -1, as is comnds(11).

It is legal to give a value for a string which is not quoted. In this case the first non-blank token after the name (and optional size) declaration is used as the string value. This is useful when defining a string which must take on the value of a macro, for example:

```
define(character,byte)
        ....
string chrdef character
```

initializes the string chrdef to be "byte".

INCLUDE

Files may be inserted into the input stream via the "include" statement. The statement

```
include filename
```

or

```
include "filename"
```

or

```
include ’filename’
```

inserts the file found on input file "filename" into the Ratfiv input in place of the include statement. This is especially useful in inserting common blocks. For example,

```
function exampl (x)
include "comblk.cmn"
exampl = x + z
return
end
```

might translate into

```
function exampl (x)
common /comblk/ q, r, z
real q, r, z
exampl = x + z
return
end
```

The form of the include statement which uses single or double quotes around the file name is preferred, and is necessary if the file name has funny characters in it such as comma.

STATEMENT GROUPING AND NULL STATEMENTS

Ratfiv allows a group of statements to be treated as a unit by enclosing them in braces -- { and } or [and]. This is true throughout the language: wherever a single statement can be used, there could also be several enclosed in braces. For example:

```
if (x > 100)
{
  call error (...)
  err = 1
  return
}
```

If curly braces are not valid characters in the local operating system, or if you wish to use upper case only, the characters "[" and "]" may be used instead of "{" and "}" respectively.

Ratfiv also allows for null statements, most useful after "for" and "while" statements. A semi-colon alone indicates a null statement. For instance,

```
while (getlin(line, int) != EOF)
        ;
```

would read lines from a file until the end-of-file was reached and

```
for (i=1; line(i) == BLANK; i=i+1)
        ;
```

positions "i" after leading blanks in a line.

FREE-FORM INPUT

Statements may be placed anywhere on a line and several may appear on one line if they are separated by semicolons, however it is not necessary to separate statements that begin with Ratfiv keywords with a semicolon. No semicolon is needed at the end of each line because Ratfiv assumes there is one statement per line unless told otherwise. Ratfiv will, however, continue lines when it seems obvious that they are not yet done, or if the line explicitly ends with an underline ("_"). Note that the underline is not included in the Fortran output of Ratfiv. Do not attempt to use a continuation character in column 6 to continue a line.

Any statement that begins with an all-numeric field is assumed to be a Fortran label and is placed in columns 1-5 upon output. Ratfiv generates labels starting at 2000 and increasing by intervals of 10, so try to use labels under 2000 for your own uses.

Statements may be passed through the Ratfiv compiler unaltered in two ways; if the toggle character "%" appears on a line by itself before and after the lines desired to be literal, the lines will be passed through unaltered to Fortran. However if the line on which the "%" character appears has other non-blank characters after the "%" character, those characters will be passed to Fortran and a matching "%" character is not needed. This is a convenient way to pass regular Fortran code through the Ratfiv compiler. Note that the "%" at the beginning of the line does not count as a Fortran column, so that code must be indented 6 spaces after the "%", or a tab character may be used if it is legal with your Fortran.

COMMENTS

A sharp character "#" in a line marks the beginning of a comment and the rest of the line is considered to be that comment. Comments and code can co-exist on the same line. For example,

```
function dummy (x)
# I made up this function to show some comments
dummy = x       #I am simply returning the parameter
return
end
```

CHARACTER TRANSLATION

Sometimes the characters >, <=, etc. are easier to read in Fortran condition statements than the standard Fortran .EQ., .LT., etc. Ratfiv allows either convention. If the special characters are used, they are translated in the following manner:

```
==              .EQ.
!= or ^= or ~=  .NE.
<               .LT.
>               .GT.
<=              .LE.
>=              .GE.
| or           .OR.
&               .AND.
! or ^ or ~     .NOT.
```

For example,

```
for (i = 1; i <= 5; i = i+1)
        ...

if (j != 100)
        ...
```

SPECIFYING NUMBERS IN BASES OTHER THAN BASE TEN

Numbers may be specified in any of the bases 2-36. Base ten is the default base. Numbers in other bases are specified as n%dddd... where 'n' is a decimal number indicating the base and 'dddd...' are digits in that base. Digits above 9 are specified by the letters a-z (A-Z), where For example,

16%ff stands for 255 base 10
8%100000 stands for -32768 base 10 on a PDP11
8%100000 stands for +32768 base 10 on a VAX

QUOTED CHARACTER STRINGS

Character strings may be enclosed in single or double quotes. To specify a single quote inside a single quoted string, repeat the single quote twice, for example:

```
write(5, (' can''t open file'))
```

Similarly, to include a double quote character inside a double quoted string, place two double quotes in a row within the string.

Octal constants which are defined by a preceding double quote as in DEC Fortran will be passed successfully through Ratfiv if only one octal constant appears per line; otherwise Ratfiv will assume that a quoted string was desired. It is recommended that Ratfiv's method for specifying bases other than decimal be used, however (see above).

Similarly, direct access read and write statements which have a single quote before the record number will also be successfully passed through Ratfiv if only one single quote appears on the line. For example,

```
write(5'n) x, y, z
```

is passed to Fortran as

```
WRITE(5'N) X, Y, Z
```

however

```
write(5'n, ('ATOM '3f10.5)) x, y, z
```

will not be correctly interpreted because of the mismatched quotes.

DEFINING MACROS

The "define" statement allows you to extend the syntax of Ratfiv with macro definitions. A macro is an alpha-numeric symbol which has a definition associated with it; whenever that symbol appears in the input, it is replaced by its definition. The replacement process is called "macro expansion". The simplest use of "define" is to define a symbolic constant; thereafter, whenever that symbolic constant occurs in the input it is replaced by the definition of that constant. "define" could be used to make these symbolic constants:

```
define(ROW,10)
define(COLUMN,25)

dimension array (ROW, COLUMN)
```

and

```
define(EOF,-10)
if (getlin(line, int) == EOF)
        ....
```

A macro such as ROW is useful because its name is more meaningful than the number "10". In addition, it is far easier to modify programs which use symbolic constants; if ROW were used consistently throughout a program, then only the definition of ROW would need to be changed when more (or less) rows are needed in "array".

Definitions may be included anywhere in the code, as long as they appear before the macro is referenced. Defined names may contain letters, digits, and the underscore (_) and dollar ($) characters. Case is significant, so that upper case names are different from lower case names.

Macros with arguments

It is also possible to define macros with arguments. For instance:

```
define(bump,$1 = $1+1)
```

defines a macro which may be used to increment a variable by 1; when

```
bump(i)
```

appears in the input, it is expanded to:

```
i = i+1
```

The "$1" in the macro definition is a placeholder for the first argument to the macro. When the macro is expanded, all occurrences of "$1" are replaced by the actual first argument, in this case "i". Up to nine macro arguments are allowed, not counting argument 0, which refers to the macro name. Here is another example:

```
define(write_buf,if ($3 > 0)
write($1,(1x,<$3>a1)) ($2(ii),ii=1,$3))
```

defines a macro which when referenced by:

```
write_buf(5, buf, n)
```

expands to:

```
if (n > 0)
write(5, (1x,<n>a1)) (buf(ii), ii=1,n)
```

"write_buf" has three arguments; argument one is the unit number, argument two is the buffer to be written out, and argument three is the number of elements in the buffer to be written. The arguments are delimited by commas when the macro is referenced.

The following macro does a prompt to the terminal, then a read:

```
define(read_prompt,{ write(5,($1)); read(5,($2)) $3 })
```

When read_prompt is referenced by:

```
read_prompt('$Enter X, Y, and Z: ', 3F10.5, (x, y, z))
```

it expands to:

```
{ write(5,('$Enter X, Y, and Z: ')); read(5,(3F10.5)) (x, y, z) }
```

The parentheses around "x, y, z" are needed to group "x, y, z" as argument 3; otherwise it would be passed as arguments 3, 4, and 5, since the commas would act as argument separators. The braces ("{" and "}") in

the definition are not necessary in most cases, but allow read_prompt to be used after structured statements as if it was a single statement. For example:

```
                    if (unit == 5)
    read_prompt('$Output file? ', 80a1, buf)
```

Pre-defined macros

In addition to "define", there are a number of other built-in macros:

_macro(x,y) equivalent to "define(x,y)".

_undef(x) undoes the most recent definition of "x". If "x" had been defined twice, "_undef(x)" would pop "x" back to it's first definition.

_repdef(x,y) replaces the current definition of "x" (if any) with "y". Equivalent to the sequence "_undef(x) _macro(x,y)".

_incr(x) converts "x" to a number and adds one to it.

_arith(x1,op1,x2[,op2,x3,...])
performs integer arithmetic (-,+,/,*) specified by "op1", "op2", etc. on "x1", "x2", etc. Evaluation is from left to right. Up to 9 arguments (5 operands and 4 operators) may be passed to "_arith".

_len(x) returns the length of string "x". The string may have commas in it.

_substr(s,m,n) returns the substring of "s" which starts at location "m" and is "n" long. If "n" is not specified or is too large, "_substr" returns the rest of the string starting at "m". If "m" is not specified, the whole string "s" is returned.

_index(s,c) returns the index in string "s" of character "c". If _len(c) is more than 1, the index of the first character in "c" is returned. If "c" doesn't occur in "s", 0 is returned.

_ifelse(a,b,c,d)
expands "a" and "b"; if "a" equals "b" as a character string, "c" is expanded and returned, else "d" is expanded and returned. Note that "d" is not expanded if "a" equals "b", and "c" is not expanded if "a" does not equal "b". "_ifelse" may be used to check for an optional macro argument; "_ifelse($2,,A,B)" checks if argument 2 is null; if so, it expands to "A", else "B".

_include(file) equivalent to the include facility described above.

Macro arguments

There are two ways to pass an argument to a macro when it is being expanded; with and without macro expansion of the argument. For instance, the arguments to the "define" builtin macro are not expanded before they are passed to it; this allows you to redefine a macro that was previously defined. For instance if "X" had been defined as "Y" (by "define(X,Y)"), then the result of

```
define(X,Z)
```

would be to redefine "X" as "Z". If the arguments to "define" were expanded before being passed to it, then the result of "define(X,Z)" would be to define "Y" as "Z", since the expansion of "X" is "Y".

When you write a macro, you can specify whether or not each argument is expanded before it is passed to your macro. Arguments specified as "$n" are expanded before being passed to the macro (these are called "eval" arguments), while arguments specified as "%n" are not expanded before being passed to the macro (these are called "noeval" arguments). For example,

```
define(set,define(%1,$2))
```

defines a macro, "set", which is like the "define" builtin macro except that the second argument, $2, is expanded before becoming the definition of the first argument, which is NOT expanded. In most cases the expansion of the second argument at definition time makes no difference; however the macro "set" lets you define macros which may be used as macro "variables", which can communicate state information between macros. An example of communication between macros through macro variables is given below in the section titled, "Example macros".

For most purposes the eval ($) form of specifying macro arguments is preferred. Noeval arguments are particularly useful in macros which define or undefine other macros, or which test whether another macro is defined.

A given argument (argument 1 for example) may not be both an eval and a noeval type, thus "$1" and "%1" could not both appear in a macro definition.

The special arguments "$&" and "%&"

When "$&" or "%&" appear in a macro definition, they are placeholders for all the arguments which are passed to the macro, including the commas which separate them. "$&" is the eval form of the arguments, and "%&" is the noeval form. "$&" and "%&" cannot both appear in the same macro definition, and macros with "$&" or "%&" in their definitions cannot also have numeric arguments such as "$1", "%2", etc.

Example:

```
define(macro,define(%&))
```

defines "macro" as being equivalent to "define", since "define" does not expand it’s arguments and it allows commas in definitions.

Example macros

Below is an example of macros which implement "while" and "endwhile" statements. Although this is not terribly useful given that Ratfiv has the "while" statement, it does illustrate some advanced techniques for writing macros. The first macro is "push":

```
define(push,define(_stack,$1)_stack)
```

"Push" pushes a new definition on top of the old definition of a macro named "_stack", then returns the new value of "_stack". "_stack" is used here as a macro variable, ie. its function is to save information between macro invocations. Specifying "$1" instead of "%1" insures that the definition being pushed onto "_stack" is completely expanded before it is pushed.

```
define(pop,_stack _undef(_stack))
```

"Pop" returns the current value of "_stack", then undefines it back to it’s previous value. Now we can define "while" and "endwhile":

```
define(while,
      push(_incr(_stack)) if (.not.($1)) goto push(_incr(_stack)))

define(endwhile,   goto _arith(pop,-,1)
      _incr(pop)    continue)
```

With "while" and "endwhile" defined, the sequence:

```
while (x < y)
      y = y/2
endwhile
```

expands to:

```
1    if (.not. (x < y)) goto 2
          y = y/2
          goto 1
2    continue
```

Of course the Ratfiv output would look different; the above output is the macro expansion before Ratfiv converts it to Fortran.

PASSING LITERAL STRINGS THROUGH RATFIV

A string may be passed literally through Ratfiv without being expanded or interpreted as a Ratfiv keyword by enclosing it in accent characters (‘). Thus the string

‘define(X,Y)‘

is passed to the output as

define(X,Y)

and is not expanded. Accented literal strings may appear within macro definitions as well, however occurrences of argument replacement strings ($n or %n) within accents are still replaced by their corresponding arguments.

To specify a literal accent character, place two of them in a row within enclosing accent characters.

CONDITIONAL CODE EXPANSION

Code may be conditionally bypassed depending on whether or not a macro is defined. The statement

```
_ifdef(x)
```

tests if x is defined as a macro. If so, then processing continues normally until the occurrence of an _elsedef or _enddef macro. If x is not defined, then all code following the _ifdef is bypassed completely until the occurrence of an _elsedef or _enddef macro.

For example:

```
define(PDP11)

    ....

_ifdef(PDP11)

    define(INTEGER_SIZE,2)
    define(MAXINT,32767)

_elsedef

    define(INTEGER_SIZE,4)
    define(MAXINT,16%7FFFFFFF)

_enddef
```

The statements:

```
_ifnotdef(x)
```

and

```
_ifndef(x)
```

are like _ifdef, except that the following code is expanded if x is NOT defined. "_ifnotdef" is the preferred form.

Nesting of _ifdefs, _ifnotdefs, _elsedefs, and _enddefs is permitted.

USING RATFIV

Ratfiv expects one or more input files on the command line and one output file preceded by a ">". For example:

RAT FILE1.RAT FILE2.RAT >FILE.FTN

The above command would cause the files FILE1.RAT and FILE2.RAT to be compiled to FILE.FTN by Ratfiv. If the Ratfiv compiler detects an error in the Ratfiv code, an error message will be printed to the terminal and to the Fortran output file at the point where the error occurred. After pre-proccessing by Ratfiv, the Fortran compiler should be invoked, producing FILE.OBJ. Many Fortran errors cannot be detected by Ratfiv; when these occur, check the Fortran listing file for the error message. It is usually not too hard to figure out where in the Ratfiv source the error is, once it has been found in the Fortran source.

Ratfiv has several switches. Switches must be placed after the RAT command only, and may not be placed on file names. All the switches may be abbreviated to one or more letters.

The /SYMBOLS switch causes Ratfiv to look on your directory for a file named "SYMBOLS."; this file is opened and read as a prefix file to the other files on your command line. If "SYMBOLS." is not on your directory, Ratfiv looks on a system-dependent directory for the "SYMBOLS." file, which normally would have some standard defines on it, such as EOS, EOF, etc.

The /F77 switch causes Ratfiv to produce VAX Fortran 77 output instead of "standard" Fortran. The Fortran 77 output is compatible with Digital Equipment Corporation VAX Fortran 77; it is not ANSII Fortran 77, as it uses the DO WHILE and END DO statements.

The /HOLLERITH switch causes Ratfiv to put out hollerith strings instead of quoted strings. This switch is included mainly for compatibility with versions of Ratfor which put out hollerith strings. So far, this switch has been needed only when compiling routines which call DEC VAX macro routines which expect hollerith strings for their arguments instead of character descriptors. Since Ratfiv normally produces quoted strings, Fortran "open" statements and other statements which require quoted strings may be used in Ratfiv programs.

EXAMPLE

The following is a sample Ratfiv program designed to show some of the commonly-used Ratfiv statements. The routine reads through a list of files, counting the lines as it goes. The subroutines and functions which are used but not defined here are available in the Ratfiv library. The symbols such as EOF, ERR, MAXLINE, and READ are defined in the file "SYMBOLS." which is included with Ratfiv. The example will not compile correctly unless the "SYMBOLS." file is read. (See "USING RATFIV" above.)

```
# main calling routine for count

        call initr4
        call count
        call endr4
        end


# count - counts lines in files

        subroutine count
        include comblk     # this file contains a common block which
                        # contains an integer variable "linect"

        character file(FILENAMESIZE), line(MAXLINE)
        integer i, f, getarg, open, getlin, itoc


# loop through the list of files

        linect = 0
        for (i = 1; getarg(i, file, FILENAMESIZE) != EOF; i = i+1)
        {
         f = open (file, READ)        # open the file
         if (f == ERR)
         {              # file could not be located
           call putlin(file, ERROUT)
           call remark (": can't open file.")
           next
         }
         else                       # read lines from the file
           while (getlin(line, f) != EOF)
             linect = linect + 1
         call close(f)              # close the file
        }

        i = itoc(linect, line, MAXLINE) # convert linect to character
        call putlin(line, STDOUT)       # write out value of linect
        call putch(NEWLINE, STDOUT)     # flush the output buffer
        return
        end
```

BOX

The command BOX is a very convenient way to undersdand the structure of of a source program.

```
BOX EXAMPLE.RAT
```

will produce:

```
# main calling routine for count

        call initr4
        call count
        call endr4
        end


# count - counts lines in files

subroutine count
include comblk     # this file contains a common block which
                   # contains an integer variable "linect"

character file(FILENAMESIZE), line(MAXLINE)
integer i, f, getarg, open, getlin, itoc


# loop through the list of files

linect = 0
for (i = 1; getarg(i, file, FILENAMESIZE) != EOF; i = i+1)
+-------------------------------------------------------------+
| f = open (file, READ)        # open the file               |
| if (f == ERR)                                              |
| +---------------------------------------------------------+ |
| | call putlin(file, ERROUT)                               | |
| | call remark (": can't open file.")                      | |
| | next                                                    | |
| +---------------------------------------------------------+ |
| else                      # read lines from the file       |
|   while (getlin(line, f) != EOF)                           |
|     linect = linect + 1                                    |
| call close(f)               # close the file               |
+-------------------------------------------------------------+

i = itoc(linect, line, MAXLINE) # convert linect to character
call putlin(line, STDOUT)       # write out value of linect
call putch(NEWLINE, STDOUT)     # flush the output buffer
return
end
```

EXAMPLE OF FORTRAN OUTPUT

Followings are four identical subroutines, the first is the original, uncomprehensive, the second is a RATFIV structured program, the third is the fortran/77 output (ratfiv command0, the fourth the hollerith standard fortran, the fifth the boxed structure which may be published rather than the first one which has been published.

```
C////////////////////////////  S P A D    VERSION DU 01.04.83
C$OPTIONS NOLIST
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++  FSPAD1 ++
C     FORTRAN PRINCIPAL POUR APPEL DES ETAPES DE S.P.A.D.          +
C     SUBROUTINE SPAD1, LISTP.                                     +
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      SUBROUTINE SPAD1 ( Q , MOTS )
C==01.04.83
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C  PROGRAMME SELECTIONNANT LES ETAPES DE  * SPAD *  A EXECUTER      *
C     DEFINITION DES NUMEROS DE FICHIERS EN  DATA,                  *
C     ET DES UNITES DE LECTURE (LECA) ET D-IMPRESSION (IMP).        *
C     CREATION DU FICHIER (LEC) DES PARAMETRES DE COMMANDE.         *
C  L-ARGUMENT  MEMOT  SERT DE TEST POUR LA POURSUITE DU PROGRAMME   *
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      DIMENSION Q(MOTS),LETAP(25) ,KART(20)
      COMMON  / VAL / TEST
      COMMON  /ENSOR/ LEC, IMP
      DATA NDICA/1/, NDONA/2/,
     1     NDIC /8/, NDON /9/, NLEG/10/, NGUS/11/, NGRI/12/, NGRO/13/,
     2     NSAV/14/, NBAND/15/,NBFOR/16/
      DATA LETAP /4HDONN,4HDPLU,4HLILE,4HCOMP,4HCORB,
     1    4HMULT,4HMULD,4HAPLU,4HGRAP,4HECLA,4HSEMI,4HTAMI,
     2    4HGRAF,4HCLAI,4HMCRE,4HTABU,4HTRIH,4HAGRA,4HARCH,
     3    4HECRI,4HCODA,4HRECI,4HTRAN,4HLIST,4HSTOP/
C..... LECA= LECTEUR DE CARTES, IMP= IMPRIMANTE
C           LEC = FICHIER AUXILIAIRE DE COPIE DES PARAMETRES
      LECA  = 5
      LEC   = 19
      IMP   = 6
C..... NOMBRE D-ETAPES ACTIVES DANS SPAD (AJOUTER  LISTP,STOP)
      NETAP = 23 + 2
      N1    = NETAP - 1
C..... CREATION DU FICHIER DES PARAMETRES SUR LEC
      REWIND LEC
      KLIST = 0
 1111 READ (LECA,5000) (KART(J),J=1,20)
      WRITE (LEC,5000) (KART(J),J=1,20)
        IF (KART(1) .EQ. LETAP(N1))    KLIST = 1
        IF (KART(1) .EQ. LETAP(NETAP)) GO TO 1112
                               GO TO 1111
 1112 REWIND LEC
        IF (KLIST .EQ. 1)            CALL LISTP (BID)
C ..... APPEL DES ETAPES SUCCESSIVES
      REWIND LEC
      MEMOT = 0
 1113   IF ( MEMOT .EQ. 0 )          GO TO 1114
      WRITE (IMP,4000)
                               RETURN
 1114 READ(LEC,1000)  METAP , M1
```

```
      KETAP = 0
        DO 1115 I = 1,NETAP
        IF(METAP .EQ. LETAP(I) )       KETAP = I
 1115 CONTINUE
        IF (KETAP .EQ. N1)            GO TO 1114
        IF (KETAP .EQ. 0)             GO TO 100
        IF (KETAP .EQ. NETAP)         GO TO 200
C ......... CHOIX DE L ETAPE DEMANDEE
C
        GO TO (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,
     1                    19,20,21,22,23,200)       , KETAP
C
C.... LECTURE DES DONNEES ET CREATION DU FICHIER-ARCHIVE NDONA
    1 CALL DONNE (  Q, MOTS, MEMOT, NDONA  )
C
                GO TO  1113
C.... CREATION DE NLEG (TABLEAU DE CORRESP.LEGERE)
    2 CALL DPLUM ( Q, MOTS, MEMOT, NLEG, NBAND )
C
                GO TO  1113
C.... LECTURE DU DICO-ARCHIVE NDICA, CREATION DE NDIC ET NDON
    3 CALL LILEX ( Q, MOTS , MEMOT, NDICA , NDONA , NDIC , NDON ,
     1                                      NBAND, NBFOR)
                GO TO  1113
C.... COMPOSANTES PRINCIPALES. CREATION DE NGUS
    4 CALL COMPL ( Q, MOTS, MEMOT, NDIC, NDON, NGUS, NBAND , NSAV )
C
                GO TO  1113
C.... CORRESPONDANCES SIMPLES. CREATION DE NGUS
    5 CALL CORBI ( Q, MOTS, MEMOT, NDIC, NDON, NGUS, NBAND, NSAV )
C
                GO TO  1113
C.... CORRESP.MULTIPLES (MEMOIRE CENTRALE). CREATION NGUS
    6 CALL MULTC ( Q, MOTS, MEMOT, NDIC, NDON, NGUS, NBAND, NSAV )
C
                GO TO  1113
C.... CORRESP.MULTIPLES (LECTURE DIRECTE). CREATION NGUS
    7 CALL MULDI ( Q, MOTS , MEMOT, NDIC , NDON , NGUS , NBAND , NSAV )
C
                GO TO  1113
C.... CORRESP.LEGERE SUR NLEG. CREATION DE NGUS
    8 CALL APLUM ( Q, MOTS, MEMOT, NLEG, NGUS, NBAND, NSAV )
C
                GO TO  1113
C.... APPELS DES GRAPHIQUES (VARIABLES ET INDIVIDUS)
    9 CALL GRAPH ( Q, MOTS , MEMOT, NGUS , NBAND)
C
                GO TO  1113
C.... CLASSIFICATION (CENTRES MOBILES) PUIS ARBRE HIERARCHIQUE
   10 CALL ECLAT ( Q, MOTS, MEMOT, NDIC, NDON, NGRI, NBAND, NBFOR )
```

```
C
              GO TO  1113
C.... CLASSIFICATION SUR FACTEURS. CREATION DE NGRI
   11 CALL SEMIS ( Q, MOTS , MEMOT, NGUS , NGRI , NBFOR )
C
              GO TO  1113
C.... COUPURE DE L-ARBRE ET DESCRIPTION DES CLASSES
   12 CALL TAMIS ( Q, MOTS , MEMOT, NDIC , NDON , NGRI , NGRO ,
     1                                  NLEG , NBFOR )
              GO TO  1113
C.... GRAPHIQUES POUR LA CLASSIFICATION (CENTRES ET DENSITE)
   13 CALL GRAFK ( Q, MOTS , MEMOT, NGUS , NGRO , NDON , NBAND , NBFOR)
C
              GO TO  1113
C.... EDITION EN CLAIR DES FACTEURS (POUR COMPL ET CORBI)
   14 CALL CLAIR ( Q, MOTS, MEMOT, NDIC, NGUS )
C
              GO TO  1113
C.... REGRESSION, ANAVAR ET ANACOV
   15 CALL MCREG ( Q, MOTS, MEMOT, NDIC, NDON )
C
              GO TO  1113
C.... TABULATIONS
   16 CALL TABUL ( Q, MOTS, MEMOT,
     1          NDICA, NDONA, NDIC, NDON, NBAND, NBFOR )
              GO TO  1113
C.... TRIS-A-PLAT, HISTOGRAMMES
   17 CALL TRIHI ( Q, MOTS, MEMOT, NDIC, NDON )
              GO TO  1113
C.... CLASSIFICATION SUR UN GRAPHE
   18 CALL AGRAF ( Q, MOTS, MEMOT, NGUS,NGRO,NBAND,NSAV,NBFOR )
C
              GO TO  1113
C.... ARCHIVAGE DE COORDONNEES ET/OU CLASSIFICATIONS
   19 CALL ARCHI ( Q, MOTS, MEMOT,NDICA,NDONA,NGUS,NGRO,NBAND,NSAV )
C
              GO TO  1113
C.... GESTION DE DICTIONNAIRE
   20 CALL ECRIT ( Q, MOTS, MEMOT, NDICA )
C
              GO TO  1113
C.... APUREMENT DES VALEURS HORS-PLAGE
   21 CALL CODAJ ( Q, MOTS, MEMOT, NDICA,NDONA,NBFOR)
C
              GO TO  1113
C .... CLASSIFICATION ASCENDENTE HIERARCHIQUE
   22 CALL RECIP ( Q, MOTS, MEMOT, NGUS, NGRI, NBAND, NBFOR)
C
              GO TO  1113
C..... CREATION DES FICHIERS NDIC ET NDON A PARTIR DE NLEG
```

```
   23 CALL TRANS ( Q, MOTS, MEMOT, NLEG, NDIC, NDON )
C
                GO TO  1113
  100 WRITE (IMP,2000) METAP,M1
      RETURN
  200 WRITE (IMP,3000) METAP,M1
 1000 FORMAT (A4,A1)
 2000 FORMAT (1H ,//,35H ERREUR SUR LE NOM D ETAPE         ,A4,A1,/)
 3000 FORMAT (1H0,65X,3H** ,A4,A1,3H **/1H0,62X,
    1       16HFIN DE L-ANALYSE /1H0,130(1H-) )
 4000 FORMAT (1H1,131(1H-)//,30X,33HERREUR FATALE : DEFAUT DE MEMOIRE)
 5000 FORMAT (20A4)
      RETURN
      END
```

```
#////////////////////////////  s p a d    version du 01.04.83
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++  fspad1 ++
#     fortran principal pour appel des etapes de s.p.a.d.          +
#     subroutine spad1, listp.                                     +
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

subroutine spad1 ( q , mots )

#==13.02.84  Version ecrite en RATFOR (Rationnal Fortran
# de Software-Tools via Ratfiv de Institute of Cancer Research)
#  par N.Brouard (Institut National d'Etudes Demographiques)

# * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
#  programme selectionnant les etapes de  * spad *  a executer      *
#     definition des numeros de fichiers en  data,                  *
#     et des unites de lecture (leca) et d-impression (imp).        *
#     creation du fichier (lec) des parametres de commande.         *
#  l-argument  memot  sert de test pour la poursuite du programme   *
# * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
define (YES,1)
define (NO,0)
define (MAXA4READER,33)
define (ASSEZ_MEMOIRE,0)
define (ETAPE_INCONNUE,0)

dimension q(mots),letap(25) ,kart(MAXA4READER)

common  / val / test
common  /ensor/ lec, imp

data ndica/1/, ndona/2/,
  ndic /8/, ndon /9/, nleg/10/, ngus/11/, ngri/12/, ngro/13/,
  nsav/14/, nband/15/,nbfor/16/

data letap /4hdonn,4hdplu,4hlile,4hcomp,4hcorb,
4hmult,4hmuld,4haplu,4hgrap,4hecla,4hsemi,4htami,
4hgraf,4hclai,4hmcre,4htabu,4htrih,4hagra,4harch,
4hecri,4hcoda,4hreci,4htran,4hlist,4hstop/

leca  = 5  #..... leca= lecteur de cartes
lec   = 19 #          lec = fichier auxiliaire de copie des parametres
imp   = 6  #          imp = imprimante
netap = 23 + 2   #..... nombre d-etapes actives dans spad (ajouter  listp,stop)
n1    = netap - 1

rewind lec #..... creation du fichier des parametres sur lec
klist = NO

repeat
{
```

```
  read (leca,500) (kart(j),j=1,MAXA4READER)
  write (lec,500) (kart(j),j=1,MAXA4READER)
  if (kart(1) == letap(n1))    klist = YES

}
until (kart(1) == letap(netap))

rewind lec
if (klist == YES)
{
  call listp (bid)
  rewind lec
}
memot = ASSEZ_MEMOIRE
list = netap - 1
ifin = netap

# ......... appel des etapes successives
for( ; ; )
{
  read(lec,100)  metap , m1
  ketap = 0
  do  i = 1,netap
  {
    if(metap .eq. letap(i) )       ketap = i
  }

  # Interruption de la boucle des etapes
  if(ketap == ETAPE_INCONNUE |  memot != ASSEZ_MEMOIRE | ketap == ifin) break
  if(ketap == list ) next

  switch ( ketap)       # ......... choix de l etape demandee
  {
        #.... lecture des donnees et creation du fichier-archive ndon
    case 1: call donne (  q, mots, memot, ndona  )
        #.... creation de nleg (tableau de corresp.legere)
    case 2: call dplum ( q, mots, memot, nleg, nband )
        #.... lecture du dico-archive ndica, creation de ndic et ndon
    case 3: call lilex ( q, mots , memot, ndica , ndona , ndic , ndon ,
                                       nband, nbfor )
        #.... composantes principales. creation de ngus
    case 4: call compl ( q, mots, memot, ndic, ndon, ngus, nband , nsav)
        #.... correspondances simples. creation de ngus
    case 5: call corbi ( q, mots, memot, ndic, ndon, ngus, nband, nsav )
        #.... corresp.multiples (memoire centrale). creation ngus
    case 6: call multc ( q, mots, memot, ndic, ndon, ngus, nband, nsav )
        #.... corresp.multiples (lecture directe). creation ngus
    case 7: call muldi ( q, mots , memot, ndic , ndon , ngus , nband , nsav)
        #.... corresp.legere sur nleg. creation de ngus
    case 8: call aplum ( q, mots, memot, nleg, ngus, nband, nsav )
```

```
        #.... appels des graphiques (variables et individus)
    case 9: call graph ( q, mots , memot, ngus , nband)
        #.... classification (centres mobiles) puis arbre hierarchique
    case 10: call eclat ( q, mots, memot, ndic, ndon, ngri, nband, nbfor )
        #.... classification sur facteurs. creation de ngri
    case 11: call semis ( q, mots , memot, ngus , ngri , nbfor )
        #.... coupure de l-arbre et description des classes
    case 12: call tamis ( q, mots , memot, ndic , ndon , ngri , ngro ,
                                    nleg , nbfor )
        #.... graphiques pour la classification (centres et densite)
    case 13: call grafk ( q, mots , memot, ngus , ngro , ndon , nband ,nbfor)
        #.... edition en clair des facteurs (pour compl et corbi)
    case 14: call clair ( q, mots, memot, ndic, ngus )
        #.... regression, anavar et anacov
    case 15: call mcreg ( q, mots, memot, ndic, ndon )
        #.... tabulations
    case 16: call tabul ( q, mots, memot,
          ndica, ndona, ndic, ndon, nband, nbfor )
        #.... tris-a-plat, histogrammes
    case 17: call trihi ( q, mots, memot, ndic, ndon )
        #.... classification sur un graphe
    case 18: call agraf ( q, mots, memot, ngus,ngro,nband,nsav,nbfor )
        #.... archivage de coordonnees et/ou classifications
    case 19: call archi ( q, mots, memot,ndica,ndona,ngus,ngro,nband,nsav)
        #.... gestion de dictionnaire
    case 20: call ecrit ( q, mots, memot, ndica )
        #.... apurement des valeurs hors-plage
    case 21: call codaj ( q, mots, memot, ndica,ndona,nbfor)
        #.... classification ascendente hierarchique
    case 22: call recip ( q, mots, memot, ngus, ngri, nband, nbfor)
        #..... creation des fichiers ndic et ndon a partir de nleg
    case 23: call trans ( q, mots, memot, nleg, ndic, ndon )
   }
}

if( ketap == ifin ) write (imp,(1h0,65x,'** ',a4,a1,'** '/1h0,62x,
    'fin de l-analyse '/1h0,130('-') ))
else if( memot != ASSEZ_MEMOIRE)
  write(imp,(1h1,131('-')//,30x,'erreur fatale : defaut de memoire')) metap, m1
else
  write(imp,(1h ,//," erreur sur le nom d etape         ",a4,a1,/)   )metap, m1

100 format (a4,a1)

500 format (MAXA4READER a4)

return
end
```

```
C////////////////////////////  s p a d    version du 01.04.83
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++  fspad1 ++
C     fortran principal pour appel des etapes de s.p.a.d.           +
C     subroutine spad1, listp.                                      +
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      SUBROUTINE SPAD1(Q, MOTS)
C==13.02.84  Version ecrite en RATFOR (Rationnal Fortran
C de Software-Tools via Ratfiv de Institute of Cancer Research)
C  par N.Brouard (Institut National d'Etudes Demographiques)
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C  programme selectionnant les etapes de  * spad *  a executer      *
C     definition des numeros de fichiers en  data,                  *
C     et des unites de lecture (leca) et d-impression (imp).        *
C     creation du fichier (lec) des parametres de commande.         *
C  l-argument  memot  sert de test pour la poursuite du programme   *
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      DIMENSION Q(MOTS), LETAP(25), KART(33)
      COMMON/VAL/TEST
      COMMON/ENSOR/LEC, IMP
      DATA NDICA/1/, NDONA/2/, NDIC/8/, NDON/9/, NLEG/10/, NGUS/11/, NGR
     *I/12/, NGRO/13/, NSAV/14/, NBAND/15/, NBFOR/16/
      DATA LETAP/4HDONN, 4HDPLU, 4HLILE, 4HCOMP, 4HCORB, 4HMULT, 4HMULD,
     * 4HAPLU, 4HGRAP, 4HECLA, 4HSEMI, 4HTAMI, 4HGRAF, 4HCLAI, 4HMCRE, 4
     *HTABU, 4HTRIH, 4HAGRA, 4HARCH, 4HECRI, 4HCODA, 4HRECI, 4HTRAN, 4HL
     *IST, 4HSTOP/
C..... leca= lecteur de cartes
      LECA = 5
C           lec = fichier auxiliaire de copie des parametres
      LEC = 19
C           imp = imprimante
      IMP = 6
C..... nombre d-etapes actives dans spad (ajouter  listp,stop)
      NETAP = 23 + 2
      N1 = NETAP - 1
C..... creation du fichier des parametres sur lec
      REWIND LEC
      KLIST = 0
2000  CONTINUE
        READ(LECA, 500) (KART(J), J = 1, 33)
        WRITE(LEC, 500) (KART(J), J = 1, 33)
        IF (.NOT.(KART(1) .EQ. LETAP(N1))) GOTO 2030
          KLIST = 1
2030    CONTINUE
2010  IF (.NOT.(KART(1) .EQ. LETAP(NETAP))) GOTO 2000
      REWIND LEC
      IF (.NOT.(KLIST .EQ. 1)) GOTO 2050
        CALL LISTP(BID)
        REWIND LEC
2050  CONTINUE
      MEMOT = 0
```

```
      LIST = NETAP - 1
      IFIN = NETAP
C ..... appel des etapes successives
2070  CONTINUE
       READ(LEC, 100) METAP, M1
       KETAP = 0
       DO 2100 I = 1, NETAP
         IF (.NOT.(METAP.EQ.LETAP(I))) GOTO 2120
           KETAP = I
2120      CONTINUE
C Interruption de la boucle des etapes
2100    CONTINUE
       IF (.NOT.(KETAP .EQ. 0 .OR. MEMOT .NE. 0 .OR. KETAP .EQ. IFIN))
     *GOTO 2140
         GOTO 2090
2140    CONTINUE
       IF (.NOT.(KETAP .EQ. LIST)) GOTO 2160
         GOTO 2080
2160    CONTINUE
       I2180 = (KETAP)
       GOTO 2180
C ..... choix de l etape demandee
C.... lecture des donnees et creation du fichier-archive ndon
2200    CONTINUE
         CALL DONNE(Q, MOTS, MEMOT, NDONA)
C.... creation de nleg (tableau de corresp.legere)
         GOTO 2190
2210    CONTINUE
         CALL DPLUM(Q, MOTS, MEMOT, NLEG, NBAND)
C.... lecture du dico-archive ndica, creation de ndic et ndon
         GOTO 2190
2220    CONTINUE
         CALL LILEX(Q, MOTS, MEMOT, NDICA, NDONA, NDIC, NDON, NBAND, NB
     *FOR)
C.... composantes principales. creation de ngus
         GOTO 2190
2230    CONTINUE
         CALL COMPL(Q, MOTS, MEMOT, NDIC, NDON, NGUS, NBAND, NSAV)
C.... correspondances simples. creation de ngus
         GOTO 2190
2240    CONTINUE
         CALL CORBI(Q, MOTS, MEMOT, NDIC, NDON, NGUS, NBAND, NSAV)
C.... corresp.multiples (memoire centrale). creation ngus
         GOTO 2190
2250    CONTINUE
         CALL MULTC(Q, MOTS, MEMOT, NDIC, NDON, NGUS, NBAND, NSAV)
C.... corresp.multiples (lecture directe). creation ngus
         GOTO 2190
2260    CONTINUE
         CALL MULDI(Q, MOTS, MEMOT, NDIC, NDON, NGUS, NBAND, NSAV)
```

```
C.... corresp.legere sur nleg. creation de ngus
      GOTO 2190
2270  CONTINUE
      CALL APLUM(Q, MOTS, MEMOT, NLEG, NGUS, NBAND, NSAV)
C.... appels des graphiques (variables et individus)
      GOTO 2190
2280  CONTINUE
      CALL GRAPH(Q, MOTS, MEMOT, NGUS, NBAND)
C.... classification (centres mobiles) puis arbre hierarchique
      GOTO 2190
2290  CONTINUE
      CALL ECLAT(Q, MOTS, MEMOT, NDIC, NDON, NGRI, NBAND, NBFOR)
C.... classification sur facteurs. creation de ngri
      GOTO 2190
2300  CONTINUE
      CALL SEMIS(Q, MOTS, MEMOT, NGUS, NGRI, NBFOR)
C.... coupure de l-arbre et description des classes
      GOTO 2190
2310  CONTINUE
      CALL TAMIS(Q, MOTS, MEMOT, NDIC, NDON, NGRI, NGRO, NLEG, NBFOR
     *)
C.... graphiques pour la classification (centres et densite)
      GOTO 2190
2320  CONTINUE
      CALL GRAFK(Q, MOTS, MEMOT, NGUS, NGRO, NDON, NBAND, NBFOR)
C.... edition en clair des facteurs (pour compl et corbi)
      GOTO 2190
2330  CONTINUE
      CALL CLAIR(Q, MOTS, MEMOT, NDIC, NGUS)
C.... regression, anavar et anacov
      GOTO 2190
2340  CONTINUE
      CALL MCREG(Q, MOTS, MEMOT, NDIC, NDON)
C.... tabulations
      GOTO 2190
2350  CONTINUE
      CALL TABUL(Q, MOTS, MEMOT, NDICA, NDONA, NDIC, NDON, NBAND, NB
     *FOR)
C.... tris-a-plat, histogrammes
      GOTO 2190
2360  CONTINUE
      CALL TRIHI(Q, MOTS, MEMOT, NDIC, NDON)
C.... classification sur un graphe
      GOTO 2190
2370  CONTINUE
      CALL AGRAF(Q, MOTS, MEMOT, NGUS, NGRO, NBAND, NSAV, NBFOR)
C.... archivage de coordonnees et/ou classifications
      GOTO 2190
2380  CONTINUE
      CALL ARCHI(Q, MOTS, MEMOT, NDICA, NDONA, NGUS, NGRO, NBAND, NS
```

```
     *AV)
C.... gestion de dictionnaire
          GOTO 2190
2390    CONTINUE
          CALL ECRIT(Q, MOTS, MEMOT, NDICA)
C.... apurement des valeurs hors-plage
          GOTO 2190
2400    CONTINUE
          CALL CODAJ(Q, MOTS, MEMOT, NDICA, NDONA, NBFOR)
C.... classification ascendente hierarchique
          GOTO 2190
2410    CONTINUE
          CALL RECIP(Q, MOTS, MEMOT, NGUS, NGRI, NBAND, NBFOR)
C..... creation des fichiers ndic et ndon a partir de nleg
          GOTO 2190
2420    CONTINUE
          CALL TRANS(Q, MOTS, MEMOT, NLEG, NDIC, NDON)
          GOTO 2190
2180    CONTINUE
        IF (I2180 .LT. 1 .OR. I2180 .GT. 23) GOTO 2190
        GOTO (2200,2210,2220,2230,2240,2250,2260,2270,2280,2290,2300,231
     *0,2320,2330,2340,2350,2360,2370,2380,2390,2400,2410,2420), I2180
2190    CONTINUE
2080  GOTO 2070
2090  CONTINUE
      IF (.NOT.(KETAP .EQ. IFIN)) GOTO 2430
        WRITE(IMP, 2450)
2450      FORMAT(1H0,65X,3H** ,A4,A1,3H** /1H0,62X,17Hfin de l-analyse /
     *1H0,130(1H-))
        GOTO 2440
2430  CONTINUE
      IF (.NOT.(MEMOT .NE. 0)) GOTO 2460
        WRITE(IMP, 2480) METAP, M1
2480      FORMAT(1H1,131(1H-)//,30X,33Herreur fatale : defaut de memoire
     *)
        GOTO 2440
2460  CONTINUE
        WRITE(IMP, 2490) METAP, M1
2490      FORMAT(1H,//,35H erreur sur le nom d etape         ,A4,A1,/)
2440  CONTINUE
100   FORMAT(A4, A1)
500   FORMAT(33A4)
      RETURN
      END
```

```
C////////////////////////////  s p a d    version du 01.04.83
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++  fspad1 ++
C     fortran principal pour appel des etapes de s.p.a.d.           +
C     subroutine spad1, listp.                                      +
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      SUBROUTINE SPAD1(Q, MOTS)
C==13.02.84  Version ecrite en RATFOR (Rationnal Fortran
C de Software-Tools via Ratfiv de Institute of Cancer Research)
C  par N.Brouard (Institut National d'Etudes Demographiques)
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C  programme selectionnant les etapes de  * spad *  a executer      *
C     definition des numeros de fichiers en  data,                  *
C     et des unites de lecture (leca) et d-impression (imp).        *
C     creation du fichier (lec) des parametres de commande.         *
C  l-argument  memot  sert de test pour la poursuite du programme   *
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      DIMENSION Q(MOTS), LETAP(25), KART(33)
      COMMON/VAL/TEST
      COMMON/ENSOR/LEC, IMP
      DATA NDICA/1/, NDONA/2/, NDIC/8/, NDON/9/, NLEG/10/, NGUS/11/, NGR
     *I/12/, NGRO/13/, NSAV/14/, NBAND/15/, NBFOR/16/
      DATA LETAP/4HDONN, 4HDPLU, 4HLILE, 4HCOMP, 4HCORB, 4HMULT, 4HMULD,
     * 4HAPLU, 4HGRAP, 4HECLA, 4HSEMI, 4HTAMI, 4HGRAF, 4HCLAI, 4HMCRE, 4
     *HTABU, 4HTRIH, 4HAGRA, 4HARCH, 4HECRI, 4HCODA, 4HRECI, 4HTRAN, 4HL
     *IST, 4HSTOP/
C..... leca= lecteur de cartes
      LECA = 5
C           lec = fichier auxiliaire de copie des parametres
      LEC = 19
C           imp = imprimante
      IMP = 6
C..... nombre d-etapes actives dans spad (ajouter  listp,stop)
      NETAP = 23 + 2
      N1 = NETAP - 1
C..... creation du fichier des parametres sur lec
      REWIND LEC
      KLIST = 0
2000  CONTINUE
        READ(LECA, 500) (KART(J), J = 1, 33)
        WRITE(LEC, 500) (KART(J), J = 1, 33)
        IF (KART(1) .EQ. LETAP(N1)) THEN
          KLIST = 1
        END IF
2010  IF (.NOT.(KART(1) .EQ. LETAP(NETAP))) GOTO 2000
      REWIND LEC
      IF (KLIST .EQ. 1) THEN
        CALL LISTP(BID)
        REWIND LEC
      END IF
      MEMOT = 0
```

```
      LIST = NETAP - 1
      IFIN = NETAP
C ..... appel des etapes successives
      DO WHILE (.TRUE.)
        READ(LEC, 100) METAP, M1
        KETAP = 0
        DO 2060 I = 1, NETAP
          IF (METAP.EQ.LETAP(I)) THEN
            KETAP = I
          END IF
C Interruption de la boucle des etapes
2060    CONTINUE
        IF (KETAP .EQ. 0 .OR. MEMOT .NE. 0 .OR. KETAP .EQ. IFIN) THEN
          GOTO 2050
        END IF
        IF (KETAP .EQ. LIST) THEN
          GOTO 2040
        END IF
        I2080 = (KETAP)
        GOTO 2080
C ..... choix de l etape demandee
C.... lecture des donnees et creation du fichier-archive ndon
2100    CONTINUE
          CALL DONNE(Q, MOTS, MEMOT, NDONA)
C.... creation de nleg (tableau de corresp.legere)
          GOTO 2090
2110    CONTINUE
          CALL DPLUM(Q, MOTS, MEMOT, NLEG, NBAND)
C.... lecture du dico-archive ndica, creation de ndic et ndon
          GOTO 2090
2120    CONTINUE
          CALL LILEX(Q, MOTS, MEMOT, NDICA, NDONA, NDIC, NDON, NBAND, NB
     *FOR)
C.... composantes principales. creation de ngus
          GOTO 2090
2130    CONTINUE
          CALL COMPL(Q, MOTS, MEMOT, NDIC, NDON, NGUS, NBAND, NSAV)
C.... correspondances simples. creation de ngus
          GOTO 2090
2140    CONTINUE
          CALL CORBI(Q, MOTS, MEMOT, NDIC, NDON, NGUS, NBAND, NSAV)
C.... corresp.multiples (memoire centrale). creation ngus
          GOTO 2090
2150    CONTINUE
          CALL MULTC(Q, MOTS, MEMOT, NDIC, NDON, NGUS, NBAND, NSAV)
C.... corresp.multiples (lecture directe). creation ngus
          GOTO 2090
2160    CONTINUE
          CALL MULDI(Q, MOTS, MEMOT, NDIC, NDON, NGUS, NBAND, NSAV)
C.... corresp.legere sur nleg. creation de ngus
```

```
      GOTO 2090
2170  CONTINUE
      CALL APLUM(Q, MOTS, MEMOT, NLEG, NGUS, NBAND, NSAV)
C.... appels des graphiques (variables et individus)
      GOTO 2090
2180  CONTINUE
      CALL GRAPH(Q, MOTS, MEMOT, NGUS, NBAND)
C.... classification (centres mobiles) puis arbre hierarchique
      GOTO 2090
2190  CONTINUE
      CALL ECLAT(Q, MOTS, MEMOT, NDIC, NDON, NGRI, NBAND, NBFOR)
C.... classification sur facteurs. creation de ngri
      GOTO 2090
2200  CONTINUE
      CALL SEMIS(Q, MOTS, MEMOT, NGUS, NGRI, NBFOR)
C.... coupure de l-arbre et description des classes
      GOTO 2090
2210  CONTINUE
      CALL TAMIS(Q, MOTS, MEMOT, NDIC, NDON, NGRI, NGRO, NLEG, NBFOR
     *)
C.... graphiques pour la classification (centres et densite)
      GOTO 2090
2220  CONTINUE
      CALL GRAFK(Q, MOTS, MEMOT, NGUS, NGRO, NDON, NBAND, NBFOR)
C.... edition en clair des facteurs (pour compl et corbi)
      GOTO 2090
2230  CONTINUE
      CALL CLAIR(Q, MOTS, MEMOT, NDIC, NGUS)
C.... regression, anavar et anacov
      GOTO 2090
2240  CONTINUE
      CALL MCREG(Q, MOTS, MEMOT, NDIC, NDON)
C.... tabulations
      GOTO 2090
2250  CONTINUE
      CALL TABUL(Q, MOTS, MEMOT, NDICA, NDONA, NDIC, NDON, NBAND, NB
     *FOR)
C.... tris-a-plat, histogrammes
      GOTO 2090
2260  CONTINUE
      CALL TRIHI(Q, MOTS, MEMOT, NDIC, NDON)
C.... classification sur un graphe
      GOTO 2090
2270  CONTINUE
      CALL AGRAF(Q, MOTS, MEMOT, NGUS, NGRO, NBAND, NSAV, NBFOR)
C.... archivage de coordonnees et/ou classifications
      GOTO 2090
2280  CONTINUE
      CALL ARCHI(Q, MOTS, MEMOT, NDICA, NDONA, NGUS, NGRO, NBAND, NS
     *AV)
```

```
C.... gestion de dictionnaire
      GOTO 2090
2290  CONTINUE
      CALL ECRIT(Q, MOTS, MEMOT, NDICA)
C.... apurement des valeurs hors-plage
      GOTO 2090
2300  CONTINUE
      CALL CODAJ(Q, MOTS, MEMOT, NDICA, NDONA, NBFOR)
C.... classification ascendente hierarchique
      GOTO 2090
2310  CONTINUE
      CALL RECIP(Q, MOTS, MEMOT, NGUS, NGRI, NBAND, NBFOR)
C..... creation des fichiers ndic et ndon a partir de nleg
      GOTO 2090
2320  CONTINUE
      CALL TRANS(Q, MOTS, MEMOT, NLEG, NDIC, NDON)
      GOTO 2090
2080  CONTINUE
     GOTO (2100,2110,2120,2130,2140,2150,2160,2170,2180,2190,2200,221
    *0,2220,2230,2240,2250,2260,2270,2280,2290,2300,2310,2320), I2080
2090  CONTINUE
2040  CONTINUE
    END DO
2050  CONTINUE
    IF (KETAP .EQ. IFIN) THEN
     WRITE(IMP, 2330)
2330     FORMAT(1H0,65X,'** ',A4,A1,'** '/1H0,62X,'fin de l-analyse '/1
    *H0,130('-'))
    ELSE IF (MEMOT .NE. 0) THEN
     WRITE(IMP, 2340) METAP, M1
2340     FORMAT(1H1,131('-')//,30X,'erreur fatale : defaut de memoire')
    ELSE
     WRITE(IMP, 2350) METAP, M1
2350     FORMAT(1H,//,' erreur sur le nom d etape         ',A4,A1,/)
    END IF
100   FORMAT(A4, A1)
500   FORMAT(33A4)
    RETURN
    END
```

```
#////////////////////////////  s p a d    version du 01.04.83
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++  fspad1 ++
#     fortran principal pour appel des etapes de s.p.a.d.          +
#     subroutine spad1, listp.                                     +
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

subroutine spad1 ( q , mots )

#==13.02.84  Version ecrite en RATFOR (Rationnal Fortran
# de Software-Tools via Ratfiv de Institute of Cancer Research)
#  par N.Brouard (Institut National d'Etudes Demographiques)

# * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
#  programme selectionnant les etapes de  * spad *  a executer      *
#     definition des numeros de fichiers en  data,                  *
#     et des unites de lecture (leca) et d-impression (imp).        *
#     creation du fichier (lec) des parametres de commande.         *
#  l-argument  memot  sert de test pour la poursuite du programme   *
# * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
define (YES,1)
define (NO,0)
define (MAXA4READER,33)
define (ASSEZ_MEMOIRE,0)
define (ETAPE_INCONNUE,0)

dimension q(mots),letap(25) ,kart(MAXA4READER)

common  / val / test
common  /ensor/ lec, imp

data ndica/1/, ndona/2/,
  ndic /8/, ndon /9/, nleg/10/, ngus/11/, ngri/12/, ngro/13/,
  nsav/14/, nband/15/,nbfor/16/

data letap /4hdonn,4hdplu,4hlile,4hcomp,4hcorb,
4hmult,4hmuld,4haplu,4hgrap,4hecla,4hsemi,4htami,
4hgraf,4hclai,4hmcre,4htabu,4htrih,4hagra,4harch,
4hecri,4hcoda,4hreci,4htran,4hlist,4hstop/

leca  = 5  #..... leca= lecteur de cartes
lec   = 19 #          lec = fichier auxiliaire de copie des parametres
imp   = 6  #          imp = imprimante
netap = 23 + 2   #..... nombre d-etapes actives dans spad (ajouter  listp,stop)
n1    = netap - 1

rewind lec #..... creation du fichier des parametres sur lec
klist = NO

repeat
+-------------------------------------------------------------------+
```

```
| read (leca,500) (kart(j),j=1,MAXA4READER)                          |
| write (lec,500) (kart(j),j=1,MAXA4READER)                          |
| if (kart(1) == letap(n1))    klist = YES                           |
|                                                                    |
+--------------------------------------------------------------------+
until (kart(1) == letap(netap))

rewind lec
if (klist == YES)
+--------------------------------------------------------------------+
| call listp (bid)                                                   |
| rewind lec                                                         |
+--------------------------------------------------------------------+
memot = ASSEZ_MEMOIRE
list = netap - 1
ifin = netap

# ......... appel des etapes successives
for( ; ; )
+-----------------------------------------------------------------------------+
| read(lec,100)  metap , m1                                                   |
| ketap = 0                                                                   |
| do  i = 1,netap                                                             |
| +-------------------------------------------------------------------------+ |
| | if(metap .eq. letap(i) )       ketap = i                                | |
| +-------------------------------------------------------------------------+ |
|                                                                             |
| # Interruption de la boucle des etapes                                      |
| if(ketap == ETAPE_INCONNUE |  memot != ASSEZ_MEMOIRE | ketap == ifin) break   |
| if(ketap == list ) next                                                     |
|                                                                             |
| switch ( ketap)       # ......... choix de l etape demandee                |
| +-------------------------------------------------------------------------+ |
| |       #.... lecture des donnees et creation du fichier-archive ndon     | |
| | case 1: call donne (  q, mots, memot, ndona  )                          | |
| |       #.... creation de nleg (tableau de corresp.legere)                | |
| | case 2: call dplum ( q, mots, memot, nleg, nband )                      | |
| |       #.... lecture du dico-archive ndica, creation de ndic et ndon     | |
| | case 3: call lilex ( q, mots , memot, ndica , ndona , ndic , ndon ,     | |
| |                                    nband, nbfor )                       | |
| |       #.... composantes principales. creation de ngus                   | |
| | case 4: call compl ( q, mots, memot, ndic, ndon, ngus, nband , nsav)    | |
| |       #.... correspondances simples. creation de ngus                   | |
| | case 5: call corbi ( q, mots, memot, ndic, ndon, ngus, nband, nsav )    | |
| |       #.... corresp.multiples (memoire centrale). creation ngus         | |
| | case 6: call multc ( q, mots, memot, ndic, ndon, ngus, nband, nsav )    | |
| |       #.... corresp.multiples (lecture directe). creation ngus          | |
| | case 7: call muldi ( q, mots , memot, ndic , ndon , ngus , nband , nsav) | |
| |       #.... corresp.legere sur nleg. creation de ngus                   | |
| | case 8: call aplum ( q, mots, memot, nleg, ngus, nband, nsav )          | |
```

```
| |       #.... appels des graphiques (variables et individus)              | |
| | case 9: call graph ( q, mots , memot, ngus , nband)                     | |
| |       #.... classification (centres mobiles) puis arbre hierarchique    | |
| | case 10: call eclat ( q, mots, memot, ndic, ndon, ngri, nband, nbfor )   | |
| |       #.... classification sur facteurs. creation de ngri               | |
| | case 11: call semis ( q, mots , memot, ngus , ngri , nbfor )            | |
| |       #.... coupure de l-arbre et description des classes               | |
| | case 12: call tamis ( q, mots , memot, ndic , ndon , ngri , ngro ,       | |
| |                                      nleg , nbfor )           | |
| |       #.... graphiques pour la classification (centres et densite)      | |
| | case 13: call grafk ( q, mots , memot, ngus , ngro , ndon , nband ,nbfor) | |
| |       #.... edition en clair des facteurs (pour compl et corbi)         | |
| | case 14: call clair ( q, mots, memot, ndic, ngus )                      | |
| |       #.... regression, anavar et anacov                                | |
| | case 15: call mcreg ( q, mots, memot, ndic, ndon )                      | |
| |       #.... tabulations                                                 | |
| | case 16: call tabul ( q, mots, memot,                                   | |
| |         ndica, ndona, ndic, ndon, nband, nbfor )                        | |
| |       #.... tris-a-plat, histogrammes                                   | |
| | case 17: call trihi ( q, mots, memot, ndic, ndon )                      | |
| |       #.... classification sur un graphe                                | |
| | case 18: call agraf ( q, mots, memot, ngus,ngro,nband,nsav,nbfor )       | |
| |       #.... archivage de coordonnees et/ou classifications              | |
| | case 19: call archi ( q, mots, memot,ndica,ndona,ngus,ngro,nband,nsav)   | |
| |       #.... gestion de dictionnaire                                     | |
| | case 20: call ecrit ( q, mots, memot, ndica )                           | |
| |       #.... apurement des valeurs hors-plage                            | |
| | case 21: call codaj ( q, mots, memot, ndica,ndona,nbfor)                | |
| |       #.... classification ascendente hierarchique                      | |
| | case 22: call recip ( q, mots, memot, ngus, ngri, nband, nbfor)          | |
| |       #..... creation des fichiers ndic et ndon a partir de nleg        | |
| | case 23: call trans ( q, mots, memot, nleg, ndic, ndon )                | |
| +-------------------------------------------------------------------------+ |
+---------------------------------------------------------------------------+

if( ketap == ifin ) write (imp,(1h0,65x,'** ',a4,a1,'** '/1h0,62x,
   'fin de l-analyse '/1h0,130('-') ))
else if( memot != ASSEZ_MEMOIRE)
  write(imp,(1h1,131('-')//,30x,'erreur fatale : defaut de memoire')) metap, m1
else
  write(imp,(1h ,//," erreur sur le nom d etape         ",a4,a1,/)   )metap, m1

100 format (a4,a1)

500 format (MAXA4READER a4)

return
end
```

DIAGNOSTICS

Unfortunately some error messages are not discussed here.

can’t open symbols file
: The special symbols file containing general purpose ratfiv definitions could not be opened. Possibly the user did not have access to the particular library the preprocessor expected to read.

can’t open included file
: File to be included was not found.

definition too long
: The number of characters in the name to be defined exceeded Ratfiv’s internal array size (current maximum is 500 characters per definition)

for clause too long
: the reinit clause of a for clause was was too long. This is a fatal error.

format too long
: A format specification in a read, write, encode, or decode statement was too long (current maximum is 600 characters)

illegal break
: Break did not occur inside a valid "while", "for", "do", or "repeat" loop.

illegal case or default
: a case or default statement occurred while not inside a switch statement

illegal else
: Else clause probably did not follow an "if" clause

illegal next
: "Next" did not occur inside a valid "for", "while", "do", or "repeat" loop

illegal right brace
: A right brace was found without a matching left brace

includes nested too deeply
: At the present, includes may only be nested 4 files deep, counting the current input file

invalid for clause
: The "for" clause did not contain a valid init, condition, and/or increment section

missing left paren
: A parenthesis was expected, probably in an "if" statement, but not found

missing parenthesis in condition
: A right parenthesis was expected, probably in an "if" statement, but not found

missing quote
: an expected quote was not found

missing right paren
: A right parenthesis was expected in a Fortran (as opposed to Ratfiv) statement but not found

illegal macro name
: Macro names must not begin with a digit and must contain only alphanumeric characters or the underscore (_) and dollar ($) characters

stack overflow in parser
: Statements were nested at too deep a level. Current maximum is 100 statements nested at a time. This is a fatal error.

token too long
: A token (word) in the sourcecode was too long to fit into one of Ratfiv's internal arrays. (Current maximum word size is 200 characters.)

too many characters pushed back
: The source code has illegally specified a Ratfiv command, or has used a keyword or macro in an illegal manner, and the parser has attempted but failed to make sense out of it. This is a fatal error.

too many definitions
: Ratfiv's internal arrays could not hold all the definitions.

unbalanced parentheses
: Unbalanced parentheses detected in a Fortran (as opposed to Ratfiv) statement

unexpected EOF
: An end-of-file was reached unexpectedly. Often this is caused by unmatched braces somewhere deep in the sourcecode.

warning possible label conflict
: This message is printed when the user has labeled a statement with a label in the 2000 and up range if the label is divisible by 10. Ratfiv statements are assigned in this range and a user-defined one may conflict with a Ratfiv-generated one.

file: can't open
: Ratfiv could not open an input file specified by the user.

IMPLEMENTATION

Ratfiv generates code by reading input files and translating any Ratfiv keywords into standard Fortran. Ratfiv does not know any Fortran and thus does not handle any Fortran error detection. Errors in Ratfiv keyword syntax are noted by a message to the user's terminal and to the Fortran output file along with an indication of the source line number which caused the problem.

This compiler was originally written by B. Kernighan and P. J. Plauger, with rewrites and enhancements by David Hanson and friends (U. of Arizona), Joe Sventek and Debbie Scherrer (Lawrence Berkely Laboratory), and William Wood (Institute For Cancer Research).

For information, comments, or bug reports, please contact:

William Wood
The Institute For Cancer Research
7701 Burholme Ave.
Philadelphia, Pa. 19111
(215) 728 2760

SEE ALSO

1) Kernighan, Brian W., "Ratfor--a Preprocessor for a Rational Fortran". Bell Laboratories publication. Also available from the UC Berkeley Computer Science library.

2) Kernighan, Brian W. and P. J. Plauger, "Software Tools". Addison-Wesley Publishing Company, Reading, Mass., 1976.

3) The rat4 user and design documents; the rc user document.

4) The Unix command "rc" in the Unix Manual (RC(I))