```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!                                                                         !!
!! CoCo Processor - Copyright (c) 1996 Imagine1, Inc.                     !!
!!                                                                         !!
!! Imagine1 claims that this software is not a complete implementation    !!
!! of the Conditional Compilation definition described in SC22WG5-N1192. !!
!! Permission to use, copy, modify, and distribute this software is       !!
!! freely granted, provided that this notice is preserved.                !!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!?>?? integer, parameter :: DOS=1, UNIX=2, MAC=3
!?>?? integer, parameter :: your_system = 4 ! Edit here for other
!?>?? integer :: system = DOS ! Edit not required here!  Use override.
!?>??
!?>?? logical :: commenting = .false. !.true. to remove "!?>??" and update
!?>?? if (commenting) then
!?> **********************************************************************
!?>
!?>                        CoCo Processor
!?>
!?> Periodic CoCo versions with new features and bug fixes will be
!?> sent to you if I have your email address.
!?>
!?>  creator   : David Epstein
!?>  e-mail    : david@imagine1.com
!?>  telephone : +1-541-383-4848
!?>  Web page  : http://www.imagine1.com/imagine1/
!?>
!?> HISTORY
!?> -------
!?>  (1) Simple Ccf Processor (1994 Oct)
!?>  (2) CCF 95 (1995 May)
!?>  (3) CoCo Processor (, this code!) (1996 Sep)
!?>
!?> BASIC IDEA
!?> ----------
!?> Since the Conditional Compilation language (CoCo) is a subset of
!?> Fortran, a simple CoCo processor is achieved by using a Fortran
!?> processor to execute the CoCo directives.
!?>
!?> DEFINITIONS
!?> -----------
!?>  coco1.f90: (this program!) Step #1 of a three step process.
!?>  coco3.f90: Step #3 of a three step process.
!?>  coco step2 program file: Output of Step #1 program, part of Step #2.
!?>  coco lines file: Output of Step #2.
!?>
!?> PROCESS
!?> -------
!?> 1. Compile this CoCo processor with your Fortran processor.
!?>     (a) This file, coco1.f90, is one of two required programs.
!?>     (b) Another file, coco3.f90, is the other required program.
!?>    You now have a CoCo processor.
!?> 2. Invoke coco1, the first part of this CoCo processor, and
!?>      (a) follow the prompts for required input or
!?>      (b) let the input come from the cocovars file which is
!?>          created automatically after your initial execution of
!?>          this CoCo processor.  Make sure to delete this file if
!?>          your input changes or you prefer interactive input.
!?> 3. Compile the coco step2 program file (coco2.f90 on DOS/UNIX/MAC).
!?> 4. Execute the coco step2 program to create the coco lines file.
```

```
!?> 5. The file produced in process 4. contains the line numbers for the
!?>    lines that are to be part of the post-CoCo-processing (assumingly
!?>    a Fortran processor, but possibly some other processor).
!?> 6. Execute the coco step3 program (coco3.f90) to produce desired
!?>    output.
!?>
!?> ALGORITHM
!?> ---------
!?> Turn ?? lines into Fortran lines by replacing ?? with blanks.
!?> Turn all other lines into output statements that writes the line
!?> number to the coco lines file.
!?>
!?> Special handling of CoCo variables is required due to Fortran
!?> requirement that the specification-part precedes the execution-part.
!?> All CoCo variable declaration lines are buffered until written to a
!?> module.  This module is also used to handle initialization of CoCo
!?> variables when the programmer wants to override initial values in
!?> the source (instead of modifying the source!)
!?>
!?> DESCRIPTION OF OUTPUT
!?> ---------------------
!?> The output from Step #3 shows the result of the CoCo If construct
!?> logic.  The lines in any false branches of a CoCo If construct are
!?> turned into Fortran comments by adding the three character "!?>" in
!?> columns one to three (as is seen on this very line).  The CoCo
!?> directives are also turned into Fortran comments using the same
!?> method.  THIS CAN ALL BE CHANGED by simply changing coco3.f90.
!?>
!?> DIFFERENCES BETWEEN THIS COCO AND THE COCO DEFINITION IN N1192
!?> -------------------------------------------------------------
!?> This simple CoCo processor differs from the full CoCo definition
!?> in two major areas:
!?>  (1) The Fortran INCLUDE line is not recognized in this processor.
!?>  (2) Constraints are not all checked in this processor.
!?>
!?> There are a few other restrictions with this processor, but they
!?> are not expected to be discovered at this early stage of the CoCo
!?> language standardization.  For example, the keywords INTEGER and
!?> LOGICAL are not expected to be split across lines.
!?>?? endif ! (commenting) then
!*****************************************************************************
 MODULE m_step1_utils
  PUBLIC :: WriteLine, StopIfIoError, HandleFirstWord, WriteCoCoVarsModule, &
           AddCoCoVarDeclLine, InitCoCoVarDeclLines, AppendTheProgram
  INTEGER, PRIVATE, PARAMETER :: MAX_COCO_VAR_DECL_LINES = 20
  CHARACTER(len=256), PRIVATE, DIMENSION(MAX_COCO_VAR_DECL_LINES) &
                              :: coco_var_lines
  INTEGER, PRIVATE             :: num_coco_var_decl_lines

 ! file names (len of 100 is a random selection and can be changed)
!
! Input file to CoCo
! Output file after Step #3
! Step #2 file MODULE+PROGRAM
! Output from running Step #2
  CHARACTER(len=100), PUBLIC ::  &
                      in_file,  &
                      out_file,  &
                      step2_file, &
                      lines_file,  &
                      batch_file !  !!!!  Options (runs batch mode)
! batch
```

```
! input
! program (scratch)
  INTEGER, PUBLIC, PARAMETER :: BA_UNUM=30, &
                                IN_UNUM=31, &
                                PR_UNUM=32, &
                                S2_UNUM=33    !    step2
 CONTAINS
  SUBROUTINE InitCoCoVarDeclLines()
     num_coco_var_decl_lines = 0
  ENDSUBROUTINE InitCoCoVarDeclLines

  SUBROUTINE AddCoCoVarDeclLine(line)
    CHARACTER(len=*), INTENT(in) :: line
     num_coco_var_decl_lines = num_coco_var_decl_lines + 1
     IF (num_coco_var_decl_lines > MAX_COCO_VAR_DECL_LINES) THEN
        PRINT *, "CoCo halts.  Limit of CoCo variable declaration lines"
        PRINT *, MAX_COCO_VAR_DECL_LINES," exceeded on line:"
        PRINT *, line
        PRINT *
        PRINT *, "Changing the value of MAX_COCO_VAR_DECL_LINES in your"
        PRINT *, "simple CoCo processor will correct this situation."
        STOP
     ENDIF
     coco_var_lines(num_coco_var_decl_lines) = " "//line(3:) !    remove ??
  ENDSUBROUTINE AddCoCoVarDeclLine

  SUBROUTINE WriteLine(unit_num, line)
  INTRINSIC trim
    INTEGER, INTENT(in)          :: unit_num
    CHARACTER(len=*), INTENT(in) :: line
     WRITE(unit=unit_num, fmt="(a)") trim(line)
  ENDSUBROUTINE WriteLine

  SUBROUTINE StopIfIoError(eof, unit_num, filename) !    Check IOSTAT variable.
    INTEGER, INTENT(in) :: eof, unit_num
    CHARACTER(len=*), INTENT(in) :: filename
     IF (eof > 0) THEN
        PRINT *, "An IO error has occurred on unit number ",unit_num
        PRINT *, " which is file ", filename
        PRINT *, "Processing stops"
        STOP
     ENDIF
  ENDSUBROUTINE StopIfIoError

  SUBROUTINE HandleFirstWord(line, is_decl, is_error, is_stop)
 ! Determine if the current CoCo line is a CoCo variable declaration
 ! line.  This simple CoCo processor expects the directive to fit on
 ! one line.
  INTRINSIC len_trim, char, ichar
    CHARACTER(len=*), INTENT(inout) :: line
    LOGICAL, INTENT(out)            :: is_decl !    ??INTEGER or ??LOGICAL
    LOGICAL, INTENT(out)            :: is_error
    LOGICAL, INTENT(out)            :: is_stop
    CHARACTER(len=64) :: word
    CHARACTER(len= 1) :: ch
    INTEGER, PARAMETER :: ICHAR_LITTLE_A = ichar("a"), &
                          ICHAR_BIG_A    = ichar("A")
    INTEGER :: pos, wpos, start_of_word_pos
    INTEGER :: line_len
    LOGICAL :: has_comma, has_colon, has_init
     is_decl  = .false.
     is_error = .false.
     is_stop  = .false.
     has_init = .false.
```

```
has_comma = .false.
has_colon = .false.
! skip over the ?? and the blanks
pos = 3
line_len = len_trim(line)
DO
   IF (pos>line_len .or.  line(pos:pos)/=" ") THEN
      EXIT
   ENDIF
   pos = pos + 1
ENDDO

! collect the first word
wpos = 0
start_of_word_pos = pos
pos = pos - 1
DO
   pos = pos + 1
   ch = line(pos:pos)
   IF (pos>line_len .or.  ch=="," .or.  ch==" " .or.  ch==":") THEN
      EXIT                    !    ',' for ',PARAMETER' and  ':'  for '::'
   ELSE
      wpos = wpos + 1
      IF (ch>="A" .and.  ch<="Z") THEN !    lower-case it
         ch = char(ICHAR_LITTLE_A+(ichar(ch)-ICHAR_BIG_A))
      ENDIF
      word(wpos:wpos) = ch
   ENDIF
ENDDO
SELECT CASE (word(:wpos))
CASE ("integer", "logical")
   is_decl = .true.
  ! Make sure this is not an assignment ??integer = etc
   DO
      IF (pos>line_len) THEN
         EXIT
      ELSEIF (line(pos:pos) == "&") THEN
         PRINT *, "WARNING: CoCo prefers decl on one line:"
         PRINT *, line(:line_len)
         EXIT
      ELSEIF (line(pos:pos) == ",") THEN
         has_comma = .true.
      ELSEIF (line(pos:pos) == ":") THEN
         has_colon = .true.
      ELSEIF (line(pos:pos) == "=") THEN
         IF (has_colon) THEN
            has_init = .true.
         ELSE
            is_decl = .false.
         ENDIF
         EXIT
      ELSE
      ENDIF
      pos = pos + 1
   ENDDO

   IF (is_decl .and.  has_init .and.  .not.  has_comma) THEN
      IF (line_len+12 > 132) THEN
         PRINT *, "TROUBLE: CoCo decl line cannot be > 120.  Line:"
         PRINT *, line(:line_len)
      ENDIF
      line(start_of_word_pos+7:) = &
      ",public,save"//line(start_of_word_pos+7:)
```

```
      ELSEIF (is_decl) THEN
         IF (line_len+7 > 132) THEN
            PRINT *, "TROUBLE: CoCo decl line cannot be > 125.  Line:"
            PRINT *, line(:line_len)
         ENDIF
         line(start_of_word_pos+7:) = ",public"//line(start_of_word_pos+7:)
      ENDIF
   CASE ("error")
      is_error = .true.
      IF (line_len+2 > 132) THEN
         PRINT *, "TROUBLE: CoCo error line cannot be > 130.  On line:"
         PRINT *, line(:line_len)
      ENDIF
      line(start_of_word_pos:) = "print*,"//line(start_of_word_pos+5:)
   CASE ("stop")
      is_stop = .true.
   ENDSELECT
ENDSUBROUTINE HandleFirstWord

SUBROUTINE WriteCoCoVarsModule(batch)
! Write the CoCoVars module.  CoCoVars contains all the CoCo variable
! declarations and a subroutine called CoCoInits.  CoCoInits contains
! assignments of any initial values for CoCo variables supplied either
! in the CoCo batch file or from standard input (interactive mode).
   LOGICAL, INTENT(in) :: batch
   INTEGER             :: i
   INTEGER             :: eof
! CoCo variable to be initialized
   CHARACTER(len=32) :: init_var, &
                        init_val     !    Initial value for CoCo variable
   CHARACTER(len=99) :: line
    OPEN (unit=S2_UNUM, action="write", status="replace", &
          position="rewind", iostat=eof, file=step2_file)
    CALL StopIfIoError(eof, S2_UNUM, step2_file)
    CALL WriteLine(S2_UNUM, "module CoCoVars")
    CALL WriteLine(S2_UNUM, "implicit none")
    CALL WriteLine(S2_UNUM, " public :: CoCoInits")

    DO i = 1, num_coco_var_decl_lines
       CALL WriteLine(S2_UNUM, coco_var_lines(i))
    ENDDO

    CALL WriteLine(S2_UNUM, "contains")
    CALL WriteLine(S2_UNUM, " subroutine CoCoInits()")

    init_var = "foo" !    any foo other than '0' since '0' terminates loop
    DO
       IF (init_var == "0") THEN
          EXIT
       ENDIF
       IF (batch) THEN
          READ (unit=BA_UNUM, fmt="(a)", iostat=eof) init_var
          IF (eof<0) THEN
             PRINT *, "Trying to read a line from CoCoVars file '", &
                   trim(batch_file), "'"
             PRINT *, "Error: Batch file expecting var-val pairs or 0"
             STOP
          ELSEIF (init_var(1:1) /= "0") THEN
             READ (unit=BA_UNUM, fmt="(A)", iostat=eof) init_val
             IF (eof<0) THEN
                PRINT *, "Trying to read a line from CoCoVars file '", &
                      trim(batch_file), "'"
                PRINT *, "Error: Batch file expecting var-val pairs or 0"
                STOP
```

```
          ENDIF
          line = trim(init_var) // "=" // trim(init_val)
          CALL WriteLine(S2_UNUM, line)
        ENDIF
      ELSE
        PRINT *, "Enter the name of a CoCo variable to be initialized"
        PRINT *, " (or '0' when you are done initializing):"
        READ *, init_var
        WRITE (unit=BA_UNUM, fmt="(A)") trim(init_var)
        IF (init_var /= "0") THEN
          PRINT *, "Enter the value you want for this CoCo variable: "
          READ *, init_val
          WRITE (unit=BA_UNUM, fmt="(A)") trim(init_val)
          line = trim(init_var) // "=" // trim(init_val)
          CALL WriteLine(S2_UNUM, line)
        ENDIF
      ENDIF
    ENDDO
    CALL WriteLine(S2_UNUM, " endsubroutine CoCoInits")
    CALL WriteLine(S2_UNUM, "endmodule CoCoVars")
  ENDSUBROUTINE WriteCoCoVarsModule

  SUBROUTINE AppendTheProgram()
   CHARACTER(len=256) :: line
   INTEGER :: eof
    REWIND(unit=PR_UNUM)
    CALL WriteLine(S2_UNUM, "!Program that will output *true* line numbers")
    DO
      READ(unit=PR_UNUM, fmt="(a)", iostat=eof) line
      IF (eof < 0) THEN
        EXIT
      ENDIF
      CALL WriteLine(S2_UNUM, line)
    ENDDO
  ENDSUBROUTINE AppendTheProgram
ENDMODULE m_step1_utils

PROGRAM CoCoStep1
  USE m_step1_utils
   INTEGER :: eof              !    for iostat=eof
   LOGICAL :: batch              !    TRUE if batch_file exists
   INTEGER :: line_count          !    count the input lines
   CHARACTER(len=  8) :: line_str !    line_count as a string
   CHARACTER(len=256) :: line
   LOGICAL :: decl_line !    true if ??integer or ??logical
   LOGICAL :: error_line
   LOGICAL :: stop_line

   ! set filenames
!?>?? if (system == DOS .or. system == UNIX .or. system == MAC) then
    step2_file = "coco2.f90"
    lines_file = "cocoline.txt"
    batch_file = "cocovars.txt"
!?>?? else ! system other than dos or unix
!?>??    !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!?>??    print *, "Edit below filenames for other file system"
!?>??    stop
!?>??    !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!?>    step2_file = "coco2 filename for your system"
!?>    lines_file = "cocoline filename for your system"
!?>    batch_file = "cocovars filename for your system"
!?>?? endif

    INQUIRE(file=batch_file, exist=batch)
```

```
IF (batch) THEN
   OPEN (unit=BA_UNUM, action="read", &
        status="old", position="rewind", file=batch_file)
   READ (unit=BA_UNUM, fmt="(a)", iostat=eof) in_file
   CALL StopIfIoError(eof, BA_UNUM, batch_file)
   IF (.not.(eof<0)) THEN
      READ (unit=BA_UNUM, fmt="(a)", iostat=eof) out_file
      CALL StopIfIoError(eof, BA_UNUM, batch_file)
      IF (.not.(eof<0)) THEN
      ! got in_file name and out_file name
      ELSE
         PRINT *, "Trying to read a line from CoCoVars file '", &
                  trim(batch_file), "'"
         PRINT *, "Error: Batch file expects input and output filenames"
         STOP
      ENDIF
   ELSE
      PRINT *, "Trying to read a line from CoCoVars file '", &
               trim(batch_file), "'"
      PRINT *, "Error: Batch file expects input and output filenames"
      STOP
   ENDIF
ELSE
   PRINT *, "Enter the name of the file for CoCo processing:"
   READ *, in_file
   PRINT *, "Enter the name you want for the output file:"
   READ *, out_file
   IF (in_file == out_file) THEN
      PRINT *, "FATAL: Input filename cannot equal output filename"
      STOP
   ENDIF
   OPEN (unit=BA_UNUM, action="write", status="replace", &
        position="rewind", iostat=eof, file=batch_file)
   CALL StopIfIoError(eof, BA_UNUM, batch_file)
   WRITE(unit=BA_UNUM, fmt="(A)") trim(in_file)
   WRITE(unit=BA_UNUM, fmt="(A)") trim(out_file)
ENDIF

OPEN (unit=IN_UNUM, action="read", &
     status="old", position="rewind", iostat=eof, file=in_file)
CALL StopIfIoError(eof, IN_UNUM, in_file)
OPEN (unit=PR_UNUM, action="readwrite", &
     status="scratch", position="rewind", iostat=eof)
CALL StopIfIoError(eof, PR_UNUM, "CoCo scratch file")

CALL WriteLine(PR_UNUM, "program CoCoIt")
CALL WriteLine(PR_UNUM, " use CoCoVars ! Has CoCo vars and init values")
CALL WriteLine(PR_UNUM, " implicit none")
CALL WriteLine(PR_UNUM, "  call CoCoInits() ! Init CoCo vars")
CALL WriteLine(PR_UNUM,&
"open (unit=34, action=""write"", status=""replace"", &")
CALL WriteLine(PR_UNUM,&
"    position=""rewind"", file=""" //trim(lines_file)// """)")

!?>?? if (commenting) then
!?> ************************************************************************
!?> **  Read each line of the input file looking for                      **
!?> **    1) CoCo lines                                                   **
!?> **    2) Other lines                                                  **
!?> **                                                                    **
!?> **    1) CoCo lines -                                                 **
!?> **        a) CoCo variable declarations are buffered until written    **
!?> **           to the CoCo module.                                      **
```

```
!?> **        b) All other CoCo lines are turned into Fortran lines by    **
!?> **           replacing the ?? in columns 1 and 2 with blanks.         **
!?> **    2) Other lines -                                                **
!?> **        Create an output statement for their line number.           **
!?> ***********************************************************************
!?>?? endif ! (commenting) then
    line_count = 0
    CALL InitCoCoVarDeclLines()
    DO !    until end of file
       READ (unit=IN_UNUM, fmt="(a)", iostat=eof) line
       IF (eof<0) THEN
          EXIT
       ENDIF
       line_count = line_count + 1

      ! In coco3.f90, note that DIR_CODE = "!?>"  ! len=3!
       IF (line(1:5) == "!?>??") THEN  !    remove the !?> and continue
          line = line(4:)
       ENDIF
      ! ?? is a CoCo line
       IF (line(1:2) == "??") THEN
         ! if a CoCo var declaration, save this line for the CoCo module;
         ! else replace the ?? with blanks (turn into a Fortran statement)
          CALL HandleFirstWord(line, decl_line, error_line, stop_line)
          IF (error_line) THEN
             CALL WriteLine(PR_UNUM, "write(unit=34, fmt=" "(a)" ") " "ERROR" "")
          ELSEIF (stop_line) THEN
             CALL WriteLine(PR_UNUM, "write(unit=34, fmt=" "(a)" ") " "STOP" "")
          ENDIF
          IF (decl_line) THEN
             CALL AddCoCoVarDeclLine(line)
          ELSE
             line(1:2) = "  "
             CALL WriteLine(PR_UNUM, line)
          ENDIF
       ELSE !    not a CoCo line
          WRITE(unit=line_str, fmt="(i8)") line_count
          CALL WriteLine(PR_UNUM, &
                         "write(unit=34, fmt=" "(a)" ") " ""//line_str//"" "")
       ENDIF
    ENDDO
    CALL WriteLine(PR_UNUM, "endprogram CoCoIt")

   ! Write the module file which contains all the CoCo var decls/inits
    CALL WriteCoCoVarsModule(batch)
    CALL AppendTheProgram() !    The step2 file now has MODULE + PROGRAM

   ! Close files and we are done
    CLOSE(unit=BA_UNUM, status="keep")
    CLOSE(unit=IN_UNUM, status="keep")
    CLOSE(unit=S2_UNUM, status="keep")

   ! Reminder of Step #2
    IF (.not. batch) THEN
       PRINT *
       PRINT *, "CoCo is creating ", trim(step2_file)
       PRINT *, " as the Step2 file to compile and run before coco3."
    ENDIF
 ENDPROGRAM CoCoStep1
```