Online Submission Assignment Cover Page

Name: Richard Break
ID#:
Date: 07/02/2012
Course #: CIS*3190
Course Name: Software for Legacy Systems
Instructor: Dr. Michael Wirth
Assignment #: 1
Assignment Name: Legacy Fortran
# of Pages (including this one):

	(For instructor's use) Grade:

Design Document

Legacy Structures and Features:

When updating the original danger subroutine, the first task was to update simple structures. Many of the existing Arithmetic IF statements had a simple purpose to check if a single value was valid which were replaced with simple IF statements. For example, the original statement:

“IF (ISNOW) 5,5,1”

Was changed to:

“checkSnow : if (isnow>0) then”

Both statements wanted to check the value of ‘isnow’. However, the altered code is intuitive to understand that the IF statement is checking for is greater than zero.

Other simple features that were implemented are changing all letters to lower case, changing comments starting with ‘C’ to ‘!’ and updating variable names. Changing all characters to lower case allowed for a more appealing look to the code. Likewise, changing the comments from ‘C’ to ‘!’ was to prevent the ‘C’ from looking like a label or another variable.

Changing the variable names came after redesigning the program. Since the old variable names were abstract, it would have been confusing to determine which segment of code was being modified if the variables in the updated version were different. An example of this is:

“IF (ADFM-30.) 19,16,16”

Was changed to:

“adjCheck: if (adjFuel-30.>=0) then”

It is hard to tell that ‘adjFuel’ used to be known at ‘ADFM’. Because of the imposed length of variables in Fortran77 being no longer then 6 characters, it is understandable how the original programmer was confined to short variable names. Since Fortran90, the maximum length was extended, allowing for more descriptive names to be chosen in the updated code.

Breaking up program into 3 different subroutines was also vital in order to make the code maintainable. Sectioning the ‘danger’ subroutine into more subroutines would have made the program complex for the simple task it was designed to do. The first subroutine, ‘input’ handles the user’s input. It was meant to abstract away from the original program to keep the ‘danger’ subroutine to its original design. The ‘computeGrassTimberIndex’ subroutine was designed to help simplify the ‘danger’ subroutine. This choice was made because the section of code the subroutine was created from had these similar two Arithmetic IF statements.

“   IF (WIND-14.) 21,25,25”

“19 IF (WIND-14.) 20,24,24”

The difficulty translating the original code was partially because of these lines of code. Both statements led to the same paths, with the only difference being that one statement required one more equation to be calculated before continuing. This resulted in the creation of the subroutine instead of creating a copy-paste scenario.

The difficult part of this assignment was how the equations and other statements were not intuitive. Not being able to start from scratch yielded issues, since that meant following the same decisions the original programmer made. Also, there were the jump structures (such as the GOTO and the Arithmetic IF statements) that make the old program hard to follow. Replacing those statements required thorough knowledge of the entire program in order to be sure the remainder of the program still functioned.

Comparison of Versions:

In contrast to the original program, the new FORTRAN code is longer than the pre-existing code. However, there are many different factors that affect the differences of the length of the programs. For example, the original code did not have any input or output to the terminal, unlike the new version. Fortran77 did not have a realistic way to determine if a loop ended, using GOTO’s and Arithmetic IF’s, where as the newer versions of FORTRAN required an ending statement, resulting in a coherent structure.

Since Arithmetic IF’s had the ability to branch to 3 different labels, it allowed for the reuse of pre-existing code. GOTO’s were able to jump to a label that has been previously visited or skipped, or ahead of the statement. In modern languages such as C, if there was any of code, those lines would be transferred into a function. In this situation, in the updated FORTRAN code the use of a subroutine was used in order to recycle some of the code in a more logical format.

Overall, rewriting this program in another language has its perks like reusability in other systems and more intuitive for programmers who are not used to handling jump structures like GOTO. However, if it was be to use in older systems, it might not run correctly or at all because of the restrictions of that system. Furthermore, the requirement for determining if there was a danger of fire is complex. Possibly making a wrapper function in C for using the original Danger subroutine might have been a better soloution.

```
! Author: rbreak

! ID: 0670342

! Course:CIS*3190

! Assignment: 1



!Main, W.A., “Computer calculation of fire danger”, Research Note NC-79, U.S. Dept. of

!Agriculture, (1969) Available: http://nrs.fs.fed.us/pubs/rn/rn_nc079.pdf


      program main


      real :: dry, wet, wind, buildUp

      real :: dryFac, fineFuel, adjFuel, grass, timber, fload

      real :: precip

      integer :: isnow=-1

      integer ::  iherb


      !Calls subroutine to get user input

      call input(dry, wet, isnow, precip, wind, buildUp, iherb)


      call danger (dry, wet, isnow, precip, wind, buildUp, iherb,

     /  dryFac, fineFuel, adjFuel, grass, timber, fload)



      write (*,*) 'Fine Fuel Moisture     = ', fineFuel

      write (*,*) 'Adjusted Fuel Moisture = ', adjFuel
```

```
    write (*,*) 'Fine Fuel Spread       = ', grass
    write (*,*) 'Timber Spread Index    = ', timber
    write (*,*) 'Fire Load Index        = ', fload
    write (*,*) 'Build Up Index         = ', buildUp

    end program main




   subroutine input(dry, wet, isnow, precip, wind, buildUp, iherb)
!Subroutine for reading in input from the user.
!dry-bulb and wet-bulb readings                 dry,wet (respectively)
!a yes-no decision regarding snow on the ground  snow
!the preceding 24-hour precipitation            precip
!the current windspeed                          wind
!yesterday's build-up index                     buildUp
!the current iherbaceous stage of vegetation    iherb
    character :: yesno
    write(*,*) 'Please enter the dry reading'
    read(*,*) dry

    write(*,*) 'Please enter the wet reading'
    read(*,*) wet

    !loops until the yes/no statement is actully correct
    do
```

```
        write(*,*) 'Please enter if there is snow on the ground(ye
/s or no)'

        read(*,*) yesno

        if ((yesno=='yes') .or. (yesno=='y')) then
            isnow= 1
            exit
        else if ((yesno=='no') .or. (yesno=='n')) then
            isnow= 0
            exit
        else
            write(*,*) 'You did no enter yes or no, try again'
        end if
    end do

    write(*,*) 'Please enter the preceding 24-hour precipitation '
    read(*,*) precip

    write(*,*) 'Please enter the current wind speed'
    read(*,*) wind

    write(*,*) 'Please enter yesterdays build-up index'
    read(*,*) buildUp

    write(*,*) 'Please enter the current iherbaceous stage of vegetati
```

```
     /on'
      read(*,*) iherb

      end subroutine input

      subroutine danger (dry,wet,isnow,precip,wind,buildUp,iherb,
     /   dryFac,fineFuel,adjFuel,grass,timber,fload)

!     routine for computing national fire danger ratings and fire load index
!     data needed for the calculations are=
!     dry,     dry bulb temperature
!     wet,     wet bulb temperature
!     snow,   some positive non zero number if there is snow on the ground
!     wind,    the current wind speed in miles per hour
!     buildUp,     the last value of the build up index
!     iherb,   the current iherb state of the district 1=cured,2=transition,3=green
!     data returned from the subroutine are
!     drying factor as                       dryFac
!     fine fuel moisture as                   fineFuel
!     adjusted (10 day lag) fuel moisture as   adjFuel
!     grass spread index will be returned as   grass
!     timber spread index will be returned as  timber
!     fire load rating (man-hour base) as      fload
!     build up index will be returned as       buildUp
```

```
!    these are the table values used in computing the danger ratings

     real,dimension(4)::a=(/-0.185900,  -0.85900,-0.059660, -0.077373/)
     real,dimension(4):: b=(/30.0, 19.2, 13.8,22.5/)
     real,dimension(3):: c=(/4.5, 12.5, 27.5/)
     real,dimension(6)::d=(/16.0, 10.0, 7.0, 5.0, 4.0, 3.0/)
     integer :: i=0
     real::difference=0.

     !Predefined values, they change based off of the input
     fineFuel= 99.
     adjFuel = 99.
     dryFac=0.
     fload=0.


!    test to see if there is snow on the ground, if there is snow then the value is greater than 0

     checkSnow : if(isnow>0) then
!    there is snow on the ground and the timber and grass spread indexes
!    must be set to zero. with a zero timber spread the fire load is
!    also zero. build up will be adjusted for precipitation.
          grass=0.
          timber=0.
          precipCheck : if  ( precip - .1 >0) then
```

```
!     precipitation exceeded    .1 inches and we reduce the build up index
                    buildUp=-50.*alog(1.-(1.-exp (-buildUp/50.))
    /               *exp (-1.175*(precip-.1)))

              end if precipCheck

              if ( buildUp<=0. ) buildUp=0.
!      buildUp being a value less than zero does not make sense.

              return
!      exits subroutine
       end if checkSnow



!     there is no snow on the ground and we will compute the spread indexes
!     and fire load
       difference=dry-wet
       do i=1,3
              if( difference - c(i) <=0) exit
       end do
       if(i>3) i=4! only occurs if and only if the do loop cycles 3 times, i should equal 4.

       fineFuel=b(i)*exp (a(i)*difference)



!     we will now find the drying factor for the day
```

```
      do i=1,6
           if ( (fineFuel - d(i)) <= 0 ) then
                cycle
           end if
            dryFac=i-1
           exit
      end do
      if(i>6) dryFac=7 !if the loop never exits early

!     test to see if the fine fuel moisture is one or less
!     if fine fuel moisture is less then one we set it to one
      if ( fineFuel-1. <0) fineFuel=1.
!     add 5 percent fine fuel moisture for each iherb stage greater than one
      fineFuel = fineFuel + ( iherb-1 ) * 5.



!     we must adjust the bui for precipitation before adding the drying factor
      if (precip -.1>0) then
!     precipitation exceeded 0.10 inches    we must reduce the
!     build up index (buildUp) by an amount equal to the rain fall
           buildUp=-50.*alog(1.-(1.-exp (-buildUp/50.))
     /     *exp (-1.175*(precip-.1)))

           if ( buildUp <0) buildUp=0.0
      end if
```

```
!     after correction for rain, if any, we are ready to add today's
!     drying factor to obtain the current build up index
      buildUp=buildUp+dryFac
!     we will adjust the grass spread index for heavy fuel lags
!     the result will be the timber spread index
!     the adjusted fuel moisture, adjFuel, adjusted for heavy fuels, will
!     now be computed
      adjFuel = .9*fineFuel +.5 +9.5*exp ( -buildUp/50.)



!     test to see if the fuel moistures are greater than 30 percent.
!     if they are, set their index values to 1.
      adjCheck: if ( adjFuel-30.>=0 ) then
          fineFuelCheck : if (  fineFuel-30.>= 0 ) then
!     fine fuel moisture is greater than 30 percent, thus we set the grass
!     and timber spread indexes to one.
              grass = 1.
              timber = 1.
              return
          end if fineFuelCheck
          timber = 1.

!     calls computeGrassTimberIndex subroutine
          call computeGrassTimberIndex(wind, grass,
```

```
     /        adjFuel,fineFuel, timber)


      else
!      fine fuel moister is less then 30 percent
             if(wind-14.<0) then
                   timber = .01312*(wind+6.)
     /             * (33.-adjFuel)**1.65 - 3.
             else
                   timber = .00918*(wind+14.)
     /             * (33.-adjFuel)**1.65 - 3.
             end if


             !calls computeGrassTimberIndex subroutine
             call computeGrassTimberIndex(wind, grass,
     /        adjFuel,fineFuel, timber)
      end if adjCheck


!     we have now computed the grass and timber spread indexes
!     of the national fire danger rating system. we have the
!     build up index and now we will compute the fire load rating



      if ( (timber<=0).or.(buildUp<=0) ) return
!     it is necessary that neither timber spread nor build up be zero
!     if either timber spread or build up is zero, fire load is zero
!     both timber spread and build up are greater than zero
```

```
      fload=1.75*alog10( timber ) + .32*alog10( buildUp ) - 1.640


!     ensure that fload is greater than zero, otherwise set it to zero.(if it is zero it doesnt need set to zero)
      if ( fload <=0)then
          fload = 0.
      else
          fload  = (10. ** fload)
       end if
      return
      end subroutine danger


      ! compute the grass and timber spread indexes
       subroutine computeGrassTimberIndex(wind, grass,
     /  adjFuel,fineFuel, timber)
          if(wind-14.<0) then
          !     test to see if the wind speed is less  than 14 mph
              grass  = .01312*(wind+6.)
     /        * (33.-fineFuel)**1.65 - 3.


              if ( timber-1. <=0) then
                  timber  = 1.
                  if ( grass-1. <0) grass = 1.
              end if
           else
           !     wind speed is greater than 14 mph. we use a different formula.
              grass  = .00918*(wind+14.)
```

```
/               * (33.- fineFuel)**1.65 - 3.


                if ( grass-99.>0 )  then

                      grass = 99.

                      if ( timber-99.>0 )timber = 99.

                end if

          end if

  end subroutine computeGrassTimberIndex
```