FILE: CALENDAR.F90

```
module calendar

   use global_env

   type calendar_node
      integer :: NU   ! node up
      integer :: NL   ! node left
      integer :: NR   ! node right
      integer :: EA   ! event info 1
      integer :: EB   ! event info 2
      integer :: EC   ! event info 3
      integer :: AR   ! event circle A (right)
      integer :: AL   ! event circle A (left)
      integer :: BR   ! event circle B (right)
      integer :: BL   ! event circle B (left)
      real(DP):: TM   ! event time
   end type calendar_node

   type(calendar_node), allocatable, dimension(:) :: event

   real(DP) :: time_current

 contains

   subroutine calendar_init(max_walkers,max_nodes)

      implicit none

      integer, intent(in) :: max_walkers, max_nodes
      integer :: i

      if(max_nodes<2*max_walkers) then
        write(fmain,*) '**calendar_init** max_nodes too small.'
      end if

      allocate(event(0:max_nodes))
      !*********************************************
      ! the root node is used as special registers.*
      !*********************************************
      event(0)%NU=0                ! Unused
      event(0)%NL=0                ! Unused
      event(0)%NR=0                ! This register contains the
         top node.
      event(0)%EA=max_walkers+1    ! The first available node in
         the pool
      event(0)%EB=0                ! Unused
      event(0)%EC=0                ! Unused
      event(0)%TM=0.0_DP           ! Unused

      !************************************************************

      ! Reset event chain for i=1,max_walkers.
                                *
      ! event(n)%AR points to the first node in chain A.
                          *
      ! event(n)%BR points to the first node in chain B.
                          *
      ! If event(n)%AR=n and event(n)%BR=n means no chain evens.
                 *
      !************************************************************
      event(1:max_walkers)%AR = (/ (i,i=1,max_walkers) /)
      event(1:max_walkers)%AL = (/ (i,i=1,max_walkers) /)
      event(1:max_walkers)%BR = (/ (i,i=1,max_walkers) /)
      event(1:max_walkers)%BL = (/ (i,i=1,max_walkers) /)

      !*********************************************************
      ! event_AR(n>max_walkers) contains a pool of empty nodes.*
      !*********************************************************
      event(max_walkers+1:max_nodes-1)%AR=(/(i+1,i=max_walkers
         +1,max_nodes-1)/)
      event(max_nodes)%AR = 0   ! This is the last node.

   end subroutine calendar_init

   subroutine calendar_close()

      deallocate(event)

   end subroutine calendar_close

!*********************************************************************


   subroutine calendar_schedule_event(action,opt1,opt2,time)

      implicit none

      integer, intent(in) :: action, opt1, opt2
      real(DP), intent(in) :: time

      integer :: node, node_new
```

```fortran
    logical :: found

    if(action<=10) then
       ! a unary event has a reserved node (node id = walker_id)
       node_new = opt1

    else
       ! a binary event and other special events need a new node from a pool.
       node_new = event(0)%EA    ! Get a empty node from a pool
       if(node_new == 0) then
          write(fmain,*) '**calendar_schedule_event**: Calendar is full.'
          stop
       end if
       event(0)%EA=event(node_new)%AR  ! Register the next empty node from a pool
                                       ! next schduling.
    end if

    if(event(0)%NR == 0) then
       ! Creating the top node.
       node = 0
       event(0)%NR = node_new

    else

       ! Inserting a new node in an appropriate place in the binary tree.
```

```
         ! Look for a node to which the new node is attached.
         found = .false.
         node = event(0)%NR  ! The top node
         do while(.not.found)
            if(time <= event(node)%TM) then
               if(event(node)%NL>0) then
                  node = event(node)%NL
               else
                  found = .true.
                  event(node)%NL = node_new
               end if
            else
               if(event(node)%NR > 0)then
                  node= event(node)%NR
               else
                  found = .true.
                  event(node)%NR = node_new
               end if
            end if
         end do
      end if

      if(action>20) then
      ! insert a new binary event into the event chains.
         event(node_new)%AR = event(opt1)%AR
         event(node_new)%AL = opt1
         event(event(opt1)%AR)%AL = node_new
         event(opt1)%AR = node_new
         event(node_new)%BR = event(opt2)%BR
         event(node_new)%BL = opt2
         event(event(opt2)%BR)%BL = node_new
         event(opt2)%BR = node_new
      end if


      event(node_new)%EA = opt1       ! event option 1  (walker_id)
      event(node_new)%EB = opt2       ! event option 2  (target id if binary event)
      event(node_new)%EC = action     ! event action
      event(node_new)%TM = time       ! envet time
      event(node_new)%NL = 0          ! Nothing below this node
      event(node_new)%NR = 0          ! Nothing below this node
      event(node_new)%NU = node       ! Node above this node


      if(debug) write(fdbg,'(a5,a7,a,i6,a,e13.5,a,i3,a,2i6)') &
        'event', 'sch: ', 'n=',node_new, ', t=',time, ', a=', action, ', o=',opt1,opt2


   end subroutine calendar_schedule_event

!**************************************************************************


   subroutine calendar_find_event(action,opt1,opt2)

      implicit none

      integer, intent(out) :: action, opt1, opt2

```

```
    integer :: node

    node = event(0)%NR    ! Satrting with the top node
                          ! Find the left most node.
    do while (event(node)%NL > 0)
       node = event(node)%NL
    end do

    ! Next event found
    opt1   = event(node)%EA
    opt2   = event(node)%EB
    action = event(node)%EC
    time_current = event(node)%TM

    if(debug) write(fdbg,'(a5,a7,a,i6,a,e13.5,a,i3,a,2i6)') &
       'event','exe: ', 'n=',node, ', t=',time_current, ', a='
          ,action, ', o=',opt1,opt2

    ! Remove the event from the event tree
    if(action<=10) then
       ! Unary event
       call calendar_delete_event(node)

    else if(action<=20) then
       ! Special event
       event(node)%AR = event(0)%EA
       event(0)%EA = node
       call calendar_delete_event(node)

    else
       ! Binary event
```

```
         call calendar_delete_event_ring(opt1)
         call calendar_delete_event_ring(opt2)

      end if

   end subroutine calendar_find_event

!***************************************************************************

   subroutine calendar_delete_event_ring(node)

      implicit none

      integer, intent(in) :: node
      integer :: next

      call calendar_delete_event(node)  ! delete unary event

      next = event(node)%AL
      do while ( next /= node )
         ! detach B-circle from A-circle
         event(event(next)%BL)%BR = event(next)%BR
         event(event(next)%BR)%BL = event(next)%BL
         call calendar_delete_event(next)    ! delete a node in A-circle
         next = event(next)%AL      ! go to next node in A-cirle
      end do

      ! Put A-circle back in the pool
      event(event(node)%AL)%AR = event(0)%EA
      event(0)%EA = event(node)%AR

      ! detach atom node from A-circle
      event(node)%AL = node
      event(node)%AR = node

      next = event(node)%BL
      do while ( next /= node )
         ! detach A-circle from B-circle
         event(event(next)%AL)%AR = event(next)%AR
         event(event(next)%AR)%AL = event(next)%AL
         call calendar_delete_event(next)  ! delete a node in B-circle

         ! Put the deleted node back in the pool
         event(next)%AR = event(0)%EA
         event(0)%EA = next
         next = event(next)%BL
      end do

      ! detach atom node from B-circle
      event(node)%BL = node
      event(node)%BR = node

   end subroutine calendar_delete_event_ring

!**************************************************************************


   subroutine calendar_delete_event(node_D)

```

```
      implicit none

      integer, intent(in) :: node_D
      integer :: node, node_L, node_R, node_U

      node_U = event(node_D)%NU
      node_R = event(node_D)%NR
      node_L = event(node_D)%NL

      if(debug) write(fdbg,'(a5,a7,a,i6,a,e13.5,a,i3,a,2i6)') &
         'event','rem: ', 'n=',node_D, ', t=',time_current, ', a
            =',event(node_D)%EC, &
         ', o=', event(node_D)%EA, event(node_D)%EB

      !There is no node below this. Do nothing.
      if (event(node_U)%NR/=node_D .and. event(node_U)%NL/=
         node_D ) return

!*****************************************************************

! Find the node to be connected to the parent of the deleted
   node
!*****************************************************************

      if ( node_R == 0 ) then
         node = node_L
      else  if( node_L == 0 ) then
         node = node_R
      else  if( event(node_R)%NL == 0) then
         node = node_R
         event(node_L)%NU = node
```

```
          event(node)%NL = node_L
       else
          node = node_R
          do while ( event(node)%NL > 0 )
             node = event(node)%NL
          end do

          event(event(node)%NU)%NL = event(node)%NR
          event(event(node)%NR)%NU = event(node)%NU
          event(node)%NR = node_R
          event(node_R)%NU = node
          event(node)%NL = node_L
          event(node_L)%NU = node

       end if

!********************
! Reconnect the trees
!********************
       if ( node /= 0 ) then
             event(node)%NU = node_U
       end if
       if( event(node_U)%NR == node_D ) then
          event(node_U)%NR = node
       else
          event(node_U)%NL = node
       end if

    end subroutine  calendar_delete_event

end module calendar
```