

FILE: CALENDAR.F90

```
1 module calendar
2
3   use global_env
4
5   type calendar_node
6       integer :: NU    ! node up
7       integer :: NL    ! node left
8       integer :: NR    ! node right
9       integer :: EA    ! event info 1
10      integer :: EB    ! event info 2
11      integer :: EC    ! event info 3
12      integer :: AR    ! event circle A (right)
13      integer :: AL    ! event circle A (left)
14      integer :: BR    ! event circle B (right)
15      integer :: BL    ! event circle B (left)
16      real(DP) :: TM    ! event time
17  end type calendar_node
18
19  type(calendar_node), allocatable, dimension(:) :: event
20
21  real(DP) :: time_current
22
23  contains
24
25  subroutine calendar_init(max_walkers,max_nodes)
26
27      implicit none
28
29      integer, intent(in) :: max_walkers, max_nodes
30      integer :: i
```

```

31
32     if(max_nodes<2*max_walkers) then
33         write(fmain,*) '**calendar_init** max_nodes too small.'
34     end if
35
36     allocate(event(0:max_nodes))
37     !*****
38     ! the root node is used as special registers.*
39     !*****
40     event(0)%NU=0                ! Unused
41     event(0)%NL=0                ! Unused
42     event(0)%NR=0                ! This register contains the
43     top node.
44     event(0)%EA=max_walkers+1    ! The first available node in
45     the pool
46     event(0)%EB=0                ! Unused
47     event(0)%EC=0                ! Unused
48     event(0)%TM=0.0_DP           ! Unused
49
50     !*****
51
52     ! Reset event chain for i=1,max_walkers.
53
54     *
55
56     ! event(n)%AR points to the first node in chain A.
57
58     *
59
60     ! event(n)%BR points to the first node in chain B.
61
62     *
63
64     ! If event(n)%AR=n and event(n)%BR=n means no chain evens.
65
66     *
67
68     !*****

```

```

54     event(1:max_walkers)%AR = (/ (i,i=1,max_walkers) /)
55     event(1:max_walkers)%AL = (/ (i,i=1,max_walkers) /)
56     event(1:max_walkers)%BR = (/ (i,i=1,max_walkers) /)
57     event(1:max_walkers)%BL = (/ (i,i=1,max_walkers) /)
58
59     !*****
60     ! event_AR(n>max_walkers) contains a pool of empty nodes.*
61     !*****
62     event(max_walkers+1:max_nodes-1)%AR=(/(i+1,i=max_walkers
        +1,max_nodes-1)/)
63     event(max_nodes)%AR = 0    ! This is the last node.
64
65 end subroutine calendar_init
66
67 subroutine calendar_close()
68
69     deallocate(event)
70
71 end subroutine calendar_close
72
73 !*****
74
75 subroutine calendar_schedule_event(action,opt1,opt2,time)
76
77     implicit none
78
79     integer, intent(in) :: action, opt1, opt2
80     real(DP), intent(in) :: time
81
82     integer :: node, node_new

```

```

83     logical :: found
84
85     if(action<=10) then
86         ! a unary event has a reserved node (node id =
            walker_id)
87         node_new = opt1
88
89     else
90         ! a binary event and other special events need a new
            node from a pool.
91         node_new = event(0)%EA      ! Get a empty node from a
            pool
92         if(node_new == 0) then
93             write(fmain,*) '**calendar_schedule_event**:'
                Calendar is full.'
94             stop
95         end if
96         event(0)%EA=event(node_new)%AR  ! Register the next
            empty node from a pool
97
98                                     ! next schduling.
99
100    end if
101
102    if(event(0)%NR == 0) then
103        ! Creating the top node.
104        node = 0
105        event(0)%NR = node_new
106
107    else
108
109        ! Inserting a new node in an appropriate place in the
            binary tree.

```

```

108      ! Look for a node to which the new node is attached.
109      found = .false.
110      node = event(0)%NR ! The top node
111      do while(.not.found)
112          if(time <= event(node)%TM) then
113              if(event(node)%NL>0) then
114                  node = event(node)%NL
115              else
116                  found = .true.
117                  event(node)%NL = node_new
118              end if
119          else
120              if(event(node)%NR > 0)then
121                  node= event(node)%NR
122              else
123                  found = .true.
124                  event(node)%NR = node_new
125              end if
126          end if
127      end do
128  end if

129
130  if(action>20) then
131      ! insert a new binary event into the event chains.
132      event(node_new)%AR = event(opt1)%AR
133      event(node_new)%AL = opt1
134      event(event(opt1)%AR)%AL = node_new
135      event(opt1)%AR = node_new
136      event(node_new)%BR = event(opt2)%BR
137      event(node_new)%BL = opt2
138      event(event(opt2)%BR)%BL = node_new

```

```

139         event (opt2) %BR = node_new
140     end if
141
142
143     event (node_new) %EA = opt1          ! event option 1  (
144         walker_id)
145     event (node_new) %EB = opt2          ! event option 2  (target
146         id if binary event)
147     event (node_new) %EC = action        ! event action
148     event (node_new) %TM = time          ! envet time
149     event (node_new) %NL = 0             ! Nothing below this node
150     event (node_new) %NR = 0             ! Nothing below this node
151     event (node_new) %NU = node          ! Node above this node
152
153     if (debug) write (fdbg, '(a5,a7,a,i6,a,e13.5,a,i3,a,2i6)') &
154         'event', 'sch: ', 'n=', node_new, ', t=', time, ', a=',
155         action, ', o=', opt1, opt2
156
157
158     end subroutine calendar_schedule_event
159
160     ! *****
161
162     subroutine calendar_find_event (action, opt1, opt2)
163
164     implicit none
165
166     integer, intent(out) :: action, opt1, opt2
167
168

```

```

166     integer :: node
167
168     node = event(0)%NR      ! Starting with the top node
169                             ! Find the left most node.
170     do while (event(node)%NL > 0)
171         node = event(node)%NL
172     end do
173
174     ! Next event found
175     opt1 = event(node)%EA
176     opt2 = event(node)%EB
177     action = event(node)%EC
178     time_current = event(node)%TM
179
180     if(debug) write(fdbg,'(a5,a7,a,i6,a,e13.5,a,i3,a,2i6)') &
181         'event','exe: ', 'n=',node, ', t=',time_current, ', a='
182         ,action, ', o=',opt1,opt2
183
184     ! Remove the event from the event tree
185     if(action<=10) then
186         ! Unary event
187         call calendar_delete_event(node)
188
189     else if(action<=20) then
190         ! Special event
191         event(node)%AR = event(0)%EA
192         event(0)%EA = node
193         call calendar_delete_event(node)
194
195     else
196         ! Binary event

```

```

196         call calendar_delete_event_ring(opt1)
197         call calendar_delete_event_ring(opt2)
198
199     end if
200
201     end subroutine calendar_find_event
202
203     !*****
204
205     subroutine calendar_delete_event_ring(node)
206
207         implicit none
208
209         integer, intent(in) :: node
210         integer :: next
211
212         call calendar_delete_event(node)    ! delete unary event
213
214         next = event(node)%AL
215         do while ( next /= node )
216             ! detach B-circle from A-circle
217             event(event(next)%BL)%BR = event(next)%BR
218             event(event(next)%BR)%BL = event(next)%BL
219             call calendar_delete_event(next)    ! delete a node in
                A-circle
220             next = event(next)%AL    ! go to next node in A-circle
221         end do
222
223         ! Put A-circle back in the pool
224         event(event(node)%AL)%AR = event(0)%EA

```



```

225     event(0)%EA = event(node)%AR
226
227     ! detach atom node from A-circle
228     event(node)%AL = node
229     event(node)%AR = node
230
231     next = event(node)%BL
232     do while ( next /= node )
233         ! detach A-circle from B-circle
234         event(event(next)%AL)%AR = event(next)%AR
235         event(event(next)%AR)%AL = event(next)%AL
236         call calendar_delete_event(next) ! delete a node in B-
            circle
237
238         ! Put the deleted node back in the pool
239         event(next)%AR = event(0)%EA
240         event(0)%EA = next
241         next = event(next)%BL
242     end do
243
244     ! detach atom node from B-circle
245     event(node)%BL = node
246     event(node)%BR = node
247
248     end subroutine calendar_delete_event_ring
249
250     ! *****
251
252     subroutine calendar_delete_event(node_D)
253

```

```

254     implicit none
255
256     integer, intent(in) :: node_D
257     integer :: node, node_L, node_R, node_U
258
259     node_U = event(node_D) %NU
260     node_R = event(node_D) %NR
261     node_L = event(node_D) %NL
262
263     if(debug) write(fdbg, '(a5,a7,a,i6,a,e13.5,a,i3,a,2i6)') &
264         'event','rem: ', 'n=',node_D, ', t=',time_current, ', a
265         =',event(node_D) %EC, &
266         ', o=', event(node_D) %EA, event(node_D) %EB
267
268     !There is no node below this. Do nothing.
269
270     if (event(node_U) %NR/=node_D .and. event(node_U) %NL/=
271         node_D ) return
272
273     ! *****
274
275     ! Find the node to be connected to the parent of the deleted
276     node
277
278     ! *****
279
280     if ( node_R == 0 ) then
281         node = node_L
282     else if( node_L == 0 ) then
283         node = node_R
284     else if( event(node_R) %NL == 0) then
285         node = node_R
286     event(node_L) %NU = node

```

```

280         event (node) %NL = node_L
281     else
282         node = node_R
283         do while ( event (node) %NL > 0 )
284             node = event (node) %NL
285         end do
286
287         event (event (node) %NU) %NL = event (node) %NR
288         event (event (node) %NR) %NU = event (node) %NU
289         event (node) %NR = node_R
290         event (node_R) %NU = node
291         event (node) %NL = node_L
292         event (node_L) %NU = node
293
294     end if
295
296     ! *****
297     ! Reconnect the trees
298     ! *****
299     if ( node /= 0 ) then
300         event (node) %NU = node_U
301     end if
302     if ( event (node_U) %NR == node_D ) then
303         event (node_U) %NR = node
304     else
305         event (node_U) %NL = node
306     end if
307
308     end subroutine  calendar_delete_event
309
310 end module calendar

```