```
  1 ! ------------------------------------------------------------------------------
  2 ! This file is part of Fortran-YAML: a lightweight YAML parser written in
  3 ! object-oriented Fortran.
  4 !
  5 ! Official repository: https://github.com/BoldingBruggeman/fortran-yaml
  6 !
  7 ! Copyright 2013-2016 Bolding & Bruggeman ApS.
  8 !
  9 ! This is free software: you can redistribute it and/or modify it under
 10 ! the terms of the GNU General Public License as published by the Free Software
 11 ! Foundation (https://www.gnu.org/licenses/gpl.html). It is distributed in the
 12 ! hope that it will be useful, but WITHOUT ANY WARRANTY; without even the
 13 ! implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 14 ! A copy of the license is provided in the COPYING file.
 15 ! ------------------------------------------------------------------------------
 16
 17 module yaml
 18
 19    use yaml_types
 20
 21    implicit none
 22
 23    private
 24
 25    public parse,error_length
 26
 27    integer,parameter :: line_length  = 2048
 28    integer,parameter :: error_length = 2048
 29
 30    type type_file
 31       integer                 :: unit   = -1
 32       character(line_length)  :: line   = ''
 33       integer                 :: indent = 0
 34       logical                 :: eof    = .false.
 35       integer                 :: iline  = 0
 36       character(error_length) :: error_message = ''
 37       logical                 :: has_error     = .false.
 38    contains
 39       procedure :: next_line
 40       procedure :: set_error
 41    end type
 42
 43 contains
 44
 45    function parse(path,unit,error) result(root)
 46       integer,               intent(in)  :: unit
 47       character(len=*),      intent(in)  :: path
 48       character(error_length),intent(out) :: error
 49       class (type_node),pointer          :: root
 50
 51       type (type_file) :: file
 52       logical          :: already_open
 53
 54       nullify(root)
 55       error = ''
 56
 57       inquire(unit=unit, opened=already_open)
 58       if (.not.already_open) open(unit=unit,file=path,status='old',action='read',err=90)
 59       file%unit = unit
 60       file%eof = .false.
 61       call file%next_line()
 62       if (.not.file%has_error) root => read_value(file)
 63       if (.not.already_open) close(file%unit)
 64       if (file%has_error) then
 65          write (error,'(a,a,i0,a,a)') trim(path),', line ',file%iline,': ',trim(file%error_message)
 66       elseif (.not.file%eof) then
 67          if (associated(root)) then
 68             select type (root)
 69                class is (type_dictionary)
 70                   write (error,'(a,a,i0,a)') trim(path),', line ',file%iline,': unexpected decrease in indentation.'
 71                class is (type_scalar)
 72                   write (error,'(a,a,i0,a)') trim(path),', line ',file%iline,': expected end of file after reading &
 73                                              &one scalar value.'
 74                class default
 75                   write (error,'(a,a,i0,a)') trim(path),', line ',file%iline,': expected end of file.'
 76             end select
 77          else
 78             write (error,'(a,a,i0,a)') trim(path),', line ',file%iline,': expected end of file.'
 79          end if
 80       end if
 81
 82       if (associated(root)) call root%set_path('')
 83
 84       return
 85
 86 90    error = 'Unable to open '//trim(path)//' for reading.'
 87    end function
 88
 89    subroutine next_line(file)
 90       class (type_file),intent(inout) :: file
 91       integer                         :: i
 92       logical                         :: done
 93
 94       done = .false.
 95       do while (.not.done)
 96          ! Read entire line
 97          read (file%unit,'(A)',end=91) file%line
 98          file%iline = file%iline + 1
```

```
 99
100         ! Determine indentation and strip this.
101         file%indent = len(file%line)
102         do i=1,len(file%line)
103            if (file%line(i:i)==achar(9)) then
104               ! Found tabs in indentation: not allowed.
105               call file%set_error('tab in indentation is not allowed.')
106               return
107            elseif (file%line(i:i)/=' ') then
108               ! Found non-space: indentation ends here.
109               file%indent = i-1
110               exit
111            end if
112         end do
113         file%line = file%line(file%indent+1:)
114
115         ! If the line starts with comment character; move to next.
116         if (file%line(1:1)=='#') cycle
117
118         ! Search for whitespace delimited comment within the string; remove if found.
119         do i=1,len_trim(file%line)-1
120            if (is_whitespace(file%line(i:i)).and.file%line(i+1:i+1)=='#') then
121               file%line = file%line(:i-1)
122               exit
123            end if
124         end do
125
126         ! Strip trailing whitespace
127         do i=len(file%line),1,-1
128            if (.not.is_whitespace(file%line(i:i))) then
129               ! We found a non-whitespace character. Strip trailing whitespace and report we have a valid line.
130               file%line = file%line(:i)
131               done = .true.
132               exit
133            end if
134         end do
135      end do
136
137      ! Check for unsupported YAML features.
138      do i=1,len_trim(file%line)
139         if (file%line(i:i)=='['.or.file%line(i:i)==']'.or.file%line(i:i)=='{'.or.file%line(i:i)=='}') then
140            call file%set_error('flow mappings and sequences using []{} are not supported.')
141            return
142         end if
143         if (file%line(i:i)=='"'.or.file%line(i:i)=='''') then
144            call file%set_error('single- and double-quoted strings are not supported.')
145            return
146         end if
147      end do
148
149      return
150
151 91   file%indent = 0
152      file%eof = .true.
153   end subroutine
154
155   recursive function read_value(file) result(node)
156      class (type_file),intent(inout) :: file
157      class (type_node),pointer       :: node
158
159      integer                    :: icolon,icolon_stop,firstindent
160      type (type_key_value_pair) :: pair
161      class (type_node), pointer :: list_item
162
163      nullify(node)
164      if (file%eof) return
165
166      if (file%line(1:2)=='- ') then
167         allocate(type_list::node)
168         firstindent = file%indent
169         do
170            file%line = file%line(3:)
171            file%indent = file%indent + 2
172            list_item => read_value(file)
173            if (file%has_error) return
174            select type (node)
175               class is (type_list)
176                  call node%append(list_item)
177            end select
178
179            ! Check indentation of next line.
180            if (file%indent>firstindent) then
181               call file%set_error('unexpected increase in indentation following list item.')
182               return
183            elseif (file%eof .or. file%indent<firstindent) then
184               ! End-of-file or decrease in indentation signifies that the list has ended.
185               return
186            end if
187         end do
188      end if
189
190      ! Find the first colon (if any)
191      call find_mapping_character(file%line,icolon,icolon_stop)
192
193      if (icolon==-1) then
194         ! No colon found: item is a value
195         allocate(type_scalar::node)
196         select type (node)
```

```
197             class is (type_scalar)
198                node%string = trim(file%line)
199          end select
200          call file%next_line()
201       else
202          ! Colon found: item starts a mapping
203          allocate(type_dictionary::node)
204          firstindent = file%indent
205          do
206             pair = read_key_value_pair(file,icolon,icolon_stop)
207             if (file%has_error) return
208             select type (node)
209                class is (type_dictionary)
210                   call node%set(pair%key,pair%value)
211             end select
212
213             ! Check indentation of next line.
214             if (file%indent>firstindent) then
215                call file%set_error('unexpected increase in indentation following key-value pair "'//trim(pair%key)//'"
    .')
216                return
217             elseif (file%eof .or. file%indent<firstindent) then
218                ! End-of-file or decrease in indentation signifies that the mapping has ended.
219                exit
220             end if
221
222             ! We are expecting a new key-value pair, since indentation has not changed. Find position of colon.
223             call find_mapping_character(file%line,icolon,icolon_stop)
224             if (icolon==-1) then
225                call file%set_error('expected a key indicated by inline ": " or trailing :')
226                return
227             end if
228          end do
229       end if
230    end function
231
232    recursive function read_key_value_pair(file,icolon,icolon_stop) result(pair)
233       class (type_file),intent(inout) :: file
234       integer,          intent(in)    :: icolon,icolon_stop
235       type (type_key_value_pair)      :: pair
236
237       integer :: istop,baseindent
238
239       istop = len_trim(file%line)
240
241       pair%key = file%line(:icolon-1)
242       if (icolon_stop==istop) then
243          ! Colon ends the line; we need to read the value from the next line.
244          baseindent = file%indent
245          call file%next_line()
246          if (file%has_error) return
247          if (file%eof .or. file%indent<baseindent .or. (file%indent==baseindent .and. file%line(1:2)/='- ')) then
248             ! Indentation equal to, or below, that of label (or file ends after label).
249             ! That implies the value of the key-value pair is null.
250             ! See YAML specification, section 7.2. Empty Nodes.
251             allocate(type_null::pair%value)
252          else
253             ! Value on next line with higher indentation - read it.
254             pair%value => read_value(file)
255          end if
256       else
257          ! Value follows colon-space. Skip the label and read the value.
258          file%line = file%line(icolon_stop+1:)
259          file%indent = file%indent + icolon_stop
260          pair%value => read_value(file)
261       end if
262    end function
263
264    subroutine find_mapping_character(string,istart,istop)
265       character(len=*),intent(in)  :: string
266       integer,         intent(out) :: istart,istop
267       integer                      :: i,length
268
269       ! Default: mapping indicator not found.
270       istart = -1
271       istop = -1
272
273       ! Search for mapping indicator
274       length = len_trim(string)
275       do i=1,length-1
276          if (string(i:i+1)==': ') then
277             ! Found "colon space" mapping indicator
278             istart = i
279             exit
280          end if
281       end do
282
283       ! No mapping indicator found yet; check whether string ends with colon.
284       if (istart==-1 .and. string(length:length)==':') istart = length
285
286       ! If we have not found a mapping indicator by now, there isn't one: return.
287       if (istart==-1) return
288
289       ! Eliminate all trailing whitespace
290       istop = istart
291       do i=istart+1,length
292          if (.not.is_whitespace(string(i:i))) then
293             istop = i-1
```

```
294             exit
295          end if
296       end do
297
298       ! Eliminate all preceding whitespace
299       do i=istart-1,1,-1
300          if (.not.is_whitespace(string(i:i))) then
301             istart = i+1
302             exit
303          end if
304       end do
305    end subroutine
306
307    logical function is_whitespace(string)
308       character(len=*),intent(in) :: string
309       ! White space in YAML includes spaces and tabs only (NB tabs are not allowed in indentation!)
310       is_whitespace = (string(1:1)==' '.or.string(1:1)==achar(9))
311    end function
312
313    subroutine set_error(file,error)
314       class (type_file),intent(inout) :: file
315       character(len=*), intent(in)    :: error
316       file%error_message = error
317       file%has_error = .true.
318    end subroutine
319
320 end module yaml
```