```fortran
MODULE KINDERREIMMOD
    IMPLICIT NONE
    PRIVATE
    PUBLIC :: CHILD_DATA, CHILD, START, BUILD_CYCLE, LAST_ONE, PUT_CYCLE, DEL_NEXT, LENGTH, PRINT_CHILD

    TYPE CHILD_DATA
        CHARACTER(LEN=10)    :: NAME
        INTEGER              :: AGE
    END TYPE

    TYPE CHILD
        TYPE(CHILD_DATA)     :: DATA
        TYPE(CHILD), POINTER :: NEXT
    END TYPE

    TYPE START
        TYPE(CHILD), POINTER :: TOP
    END TYPE

    CONTAINS

    SUBROUTINE INIT_CYCLE (S)                ! sets S into NULL state
        TYPE(START), INTENT(OUT) :: S
        NULLIFY(S%TOP)
    END SUBROUTINE

    SUBROUTINE BUILD_CYCLE(FILENAME, LIST)  ! creates an cyclic list of children (incl. reading from file)
        CHARACTER(LEN=*), INTENT(IN)  :: FILENAME
        TYPE(START), INTENT(INOUT)    :: LIST
        TYPE(CHILD_DATA)              :: READCHILD
        INTEGER                       :: ios, err

        ! put list in empty state
        CALL INIT_CYCLE(LIST)

        OPEN(UNIT=33, FILE=TRIM(FILENAME), IOSTAT=ios, ACTION="READ")

        IF (ios == 0) THEN

            ! reading children
            DO
                READ(33, *, IOSTAT=err) READCHILD

                IF (err == 0) THEN
                    CALL INS_TAIL(LIST, READCHILD)
                ELSE
                    EXIT
                END IF

            END DO

        ELSE
            WRITE(*,*) "ERROR: Cannot open the input file."
        END IF

        CLOSE(UNIT=33)

        ! connect tail with head of list
        CALL CONNECT(LIST)

    END SUBROUTINE BUILD_CYCLE

    SUBROUTINE INS_TAIL(LIST, TO_INS)       ! insert element at the end (tail) of a list
        TYPE(START), INTENT(INOUT) :: LIST
        TYPE(CHILD_DATA)           :: TO_INS
        TYPE(CHILD), POINTER       :: LIST_ELEM, TAIL

        ALLOCATE(LIST_ELEM, TAIL)

        IF (EMPTY(LIST)) THEN

            LIST_ELEM%DATA =  TO_INS        ! put child data into an element of list
            LIST%TOP       => LIST_ELEM
            NULLIFY(LIST_ELEM%NEXT)

        ELSE
            TAIL => LIST%TOP

            DO WHILE(ASSOCIATED(TAIL%NEXT))
                TAIL => TAIL%NEXT
            END DO

            LIST_ELEM%DATA = TO_INS
            TAIL%NEXT      => LIST_ELEM
            NULLIFY(LIST_ELEM%NEXT)

        END IF

    END SUBROUTINE INS_TAIL

    SUBROUTINE CONNECT(LIST)                ! connect tail with head of list
        TYPE(START), INTENT(INOUT) :: LIST
        TYPE(CHILD), POINTER       :: TAIL

        ALLOCATE(TAIL)
        TAIL => LIST%TOP

        DO WHILE (ASSOCIATED(TAIL%NEXT))
            TAIL => TAIL%NEXT
        END DO

        TAIL%NEXT => LIST%TOP

        DEALLOCATE(TAIL)
    END SUBROUTINE CONNECT

    FUNCTION LAST_ONE(LIST)                 ! checks if one child is remaining
        TYPE(START), INTENT(IN)  :: LIST
        LOGICAL                  :: LAST_ONE

        LAST_ONE = ASSOCIATED(LIST%TOP%NEXT, LIST%TOP)
    END FUNCTION LAST_ONE

    FUNCTION EMPTY(LIST)                    ! checks if the list is empty (no children)
        TYPE(START), INTENT(IN)  :: LIST
        LOGICAL                  :: EMPTY

        EMPTY = .NOT. ASSOCIATED(LIST%TOP)
    END FUNCTION EMPTY

    SUBROUTINE PUT_CYCLE(LIST,CURR_CHILD)   ! prints the list, starting with CURR_CHILD
        TYPE(START), INTENT(IN)  :: LIST
        TYPE(CHILD), POINTER     :: CURR_CHILD
        TYPE(CHILD), POINTER     :: HELP

        WRITE(*,*) "NAME: ", CURR_CHILD%DATA%NAME, " ALTER: ", CURR_CHILD%DATA%AGE
        HELP => CURR_CHILD%NEXT

        DO WHILE(.NOT. ASSOCIATED(CURR_CHILD, HELP))
            WRITE(*,*) "NAME: ", HELP%DATA%NAME, " ALTER: ", HELP%DATA%AGE
            HELP => HELP%NEXT
        END DO

    END SUBROUTINE PUT_CYCLE

    SUBROUTINE DEL_NEXT(LIST, CURR_CHILD)   ! deletes element following the CURR_CHILD
        TYPE(START), INTENT(INOUT) :: LIST
        TYPE(CHILD), POINTER       :: CURR_CHILD, HELP

        ALLOCATE(HELP)

        HELP%NEXT => CURR_CHILD%NEXT%NEXT
        IF (ASSOCIATED(CURR_CHILD%NEXT, LIST%TOP)) THEN
            LIST%TOP => HELP%NEXT
        END IF

        CURR_CHILD%NEXT => HELP%NEXT

        DEALLOCATE(HELP)

    END SUBROUTINE DEL_NEXT

    FUNCTION LENGTH(LIST)                   ! returns the length of the list
        TYPE(START), INTENT(IN) :: LIST
        TYPE(CHILD), POINTER    :: HELP
        INTEGER                 :: LENGTH

        LENGTH = 1
        HELP => LIST%TOP

        DO WHILE(.NOT. ASSOCIATED(HELP%NEXT, LIST%TOP))
            LENGTH = LENGTH + 1
            HELP => HELP%NEXT
        END DO

    END FUNCTION LENGTH

    
```

```fortran
    SUBROUTINE PRINT_CHILD(CURR_CHILD)      ! prints data of current child
        TYPE(CHILD), POINTER :: CURR_CHILD

        WRITE(*,*) "NAME: ", CURR_CHILD%DATA%NAME, ", ALTER: ", CURR_CHILD%DATA%AGE
    END SUBROUTINE PRINT_CHILD

END MODULE KINDERREIMMOD
```