```
!-------------------------------------------------------------
!>  Module Name:    EmployeeModule
!>  Author:         Blake Lucas and Luke Reddick
!
!>  This module acts as the "Employee" class that constructs
!   the linked list. It houses the methods used to
!   manipulate the linked list
!-------------------------------------------------------------

module EmployeeModule
    implicit none
    type Employee                                       ! Custom type Employee, creates the linked list
        integer id
        character(31) :: name
        character(31) :: deptName
        character(31) :: position
        real salary
        type(Employee), pointer :: next => null()
    end type Employee

    contains
    subroutine printData(emp)                           ! Prints the data for a given Employee
        type(Employee), pointer :: emp
        if (associated(emp)) then                       ! Perform null pointer check
            write(1,*), emp%name, " (", emp%id, ")"
            write(1,*), "Department: ",emp%deptName
            write(1,*), "Position: ", emp%position
            write(1,"(A8 F8.2)"), "Salary: ", emp%salary
        end if
    end subroutine printData

    subroutine printAllEmps(head)                       ! Traverses the linked list and prints all Employees
        type(Employee), pointer :: head
        type(Employee), pointer :: current
        allocate(current, source=head%next)
        write(1,*), "All Employees:"
        do while(associated(current))
            call printData(current)
            current => current%next
        end do
        write(1,*), "All Employees Displayed."
    end subroutine printAllEmps

    subroutine insert(head, emp)                        ! Inserts a new Employee into the ascending-ordered linked list
        type(Employee), pointer :: head
        type(Employee), pointer :: emp
        type(Employee), pointer :: current
        type(Employee), pointer :: previous
        type(Employee), pointer :: temp
        current => head
        write(1,*), "Inserting employee ", emp%id
        do while ( current%id < emp%id )                ! Goes while the new Employee's ID is greater than the current node
```

```
53            if ( associated(current%next) ) then
54                previous => current
55                current => current%next
56            else
57                current%next => emp
58                return
59            end if
60        end do
61        emp%next => current;                                ! Performs the insertion
62        previous%next => emp
63    end subroutine insert
64
65    subroutine delete(head, i)                              ! Deletes an Employee from the
  linked list
66        type(Employee), pointer :: head
67        type(Employee), pointer :: current
68        type(Employee), pointer :: previous
69        integer :: i
70        current => head
71        previous => head
72        do while ( associated(current%next) )
73            if (current%id == i) then
74                previous%next => current%next               ! Link around the Employee to be
  deleted
75                deallocate(current)                         ! Free current's memory
76                return
77            end if
78            previous => current                             ! Keep a record of the previous
  node
79            current => current%next                         ! Advance the iteration
80        end do
81
82        if(.not.associated(current%next) ) then             ! Informs the user if an invalid
  ID was given
83            write(1,*) "Could not find employee ", i
84        end if
85    end subroutine delete
86
87    subroutine updateLN(head, i, last)                      ! Updates the Employee's last name
88        type(Employee), pointer :: head
89        type(Employee), pointer :: current
90        integer :: i
91        character(12) :: last
92        current => findById(head, i)                        ! Gets the Employee by ID
93        if (associated(current)) current%name = last        ! Update the last name
94    end subroutine updateLN
95
96    subroutine updateTitle(head, i, title)                  ! Update the Employee's title
97        type(Employee), pointer :: head
98        type(Employee), pointer :: current
99        integer :: i
100       character(12) :: title
101       current => findById(head, i)                        ! Gets the Employee by ID
102       if (associated(current)) then
103           current%position = title                        ! Update the title
104       end if
105   end subroutine updateTitle
```

```
106
107     subroutine updateDept(head, i, dept)                    ! Updates the Employee's
    Department
108         type(Employee), pointer :: head
109         type(Employee), pointer :: current
110         integer :: i
111         character(23) :: dept
112         current => findById(head, i)                        ! Gets the Employee by ID
113         if (associated(current)) current%deptName = dept    ! Updates the department
114     end subroutine updateDept
115
116     subroutine updatePay(head, i, pay)                      ! Updates the Employee's salary
117         type(Employee), pointer :: head
118         type(Employee), pointer :: current
119         integer :: i
120         real :: pay
121         current => findById(head, i);                       ! Gets the Employee by ID
122         if (associated(current)) current%salary = pay       ! Updates the salary
123     end subroutine updatePay
124
125     subroutine printDept(head, dept)                        ! Traverses the list and prints
    all employees in a given department
126         type(Employee), pointer :: head
127         type(Employee), pointer :: current
128         character(12) :: dept
129         current => head
130         do while(associated(current%next))
131             if (current%deptName .eq. dept) then
132                 call printData(current)
133             end if
134             current => current%next
135         end do
136     end subroutine printDept
137
138     function findById(head, id) result (emp)                ! Finds and returns an Employee by
    ID
139         type(Employee), pointer :: head
140         type(Employee), pointer :: emp
141         type(Employee), pointer :: current
142         integer id
143         current => head
144         do while (current%id < id)                          ! Traverses the linked list while
    ID is less than the specified ID
145             if (associated(current%next)) then              ! Acts much like a iterative
    search
146                 current => current%next
147             else
148                 write(1,*), "Cannot find employee ", id
149                 nullify(emp)
150                 return
151             end if
152         end do
153         if (current%id == id) then
154             emp => current
155             return
156         else
157             write(1,*), "Cannot find employee ", id
```

```
158             nullify(emp)
159         end if
160     end function findById
161
162 end module EmployeeModule
163
```