Solution 9: Recursion

ETH Zurich

1 An infectious task

Solution

Variant 1

Correct. This version works and uses tail recursion. It will always set p to be infected first, and then call *infect* on his/her coworker. The recursion ends when either there is no coworker, or the coworker is already infected. The second condition is crucial because without it there would be endless recursion if the structure is cyclic.

Variant 2

Incorrect. This version results in endless recursion if the structure is cyclic. The main cause is that the coworker does not get infected before the recursive call is made, so that with a cyclic structure there will never be anybody infected to terminate the recursion.

Variant 3

Incorrect. This version results in an endless loop if the structure is cyclic. The main problem is with the loop's exit condition that does not include the case when q is already infected.

Variant 4

Correct. This version works. It will however call *set_has_flu* twice on all reachable persons except the initial one. On the initial person *set_has_flu* will be called once in case of non-circular structure and three times in case of circular structure.

Multiple coworkers

Listing 1: Class *PERSON2*

```
1 class
    PERSON2
3
  create
5   make

7 feature -- Initialization

9   make (a_name: STRING)
        -- Initialize with name.
11    require
        a_name_valid: a_name /= Void and then not a_name.is_empty
```

```
13      do
          name := a_name
15        create coworkers.make
        ensure
17        name_set: name = a_name
          coworkers_exists: coworkers /= Void
19      end

21  feature -- Access

23    name: STRING
          -- First name
25
      coworkers: LINKED_LIST [PERSON2]
27        -- Person that Current works with

29    has_flu: BOOLEAN
          -- Does he/she have the flu?
31
    feature -- Element change
33
      add_coworker (p: PERSON2)
35        -- Set 'coworker' to 'p'.
        require
37        p_exists: p /= Void
          p_different: p /= Current
39        not_has_p: not coworkers.has (p)
        do
41        coworkers.extend (p)
        ensure
43        coworker_set: coworkers.has (p)
        end
45
      set_has_flu
47        -- Set 'has_flu' to True.
        do
49        has_flu := True
        ensure
51        has_flu: has_flu
        end
53
    invariant
55
      name_valid: name /= Void and then not name.is_empty
57    coworkers_exists: coworkers /= Void

59  end
```

```
      infect (p: PERSON2)
          -- List of persons.
        require
          p_exists: p /= Void
```

```
    do
      p. set_has_flu
      from
        p.coworkers.start
      until
        p.coworkers.off
      loop
        if not p.coworkers.item.has_flu then
          infect (p.coworkers.item)
        end
        p.coworkers.forth
      end
    end
```

2 Calculating reachable stations in Paris

Solution

Listing 2: Class *RECURSIVE_HIGHLIGHTING*

```
1 indexing
    description: "Recursive highlighting class (Assignment 9)"
3   date: "$Date$"
    revision: "$Revision$"
5
  class
7   RECURSIVE_HIGHLIGHTING

9 inherit

11   TOURISM

13 feature -- Explore Paris

15   show is
        -- Highlight stations that are reachable within a certain time limit.
17     do
        Paris.display
19      wait

21       highlight_reachable_stations (station_balard, 10.0)
      end
23
     highlight_reachable_stations (a_station: TRAFFIC_STATION; t: REAL)
25      -- Highlight all stations that are reachable from 'a_station' within travel time 't'.
      require
27       t_positive: t > 0.0
      local
29      s: TRAFFIC_STOP
        i: INTEGER
31    do
        a_station.highlight
33      from
```

```
        i := 1
35      until
          i > a_station.stops.count
37      loop
          s := a_station.stops.i_th (i)
39        if s.right /= Void and then (t - s.time_to_next) >= 0.0 then
            highlight_reachable_stations (s.right.station, t - s.time_to_next)
41        end
          i := i + 1
43      end
    end
45
end
```

3 Get me out of this maze!

Solution

Listing 3: Class *MAZE_READER*

```
class
  MAZE_READER

feature

  read_maze (f: STRING)
      -- Read a maze from file with filename 'f'.
    local
      file : PLAIN_TEXT_FILE
      n, m, i: INTEGER
    do
      has_error := False
      error_message := ""
      create file.make (f)
      if not file.exists then
        has_error := True
        error_message := "File " + f.out + " does not exist.%N"
      else
        file.open_read
        if not file.is_open_read then
          has_error := True
          error_message := "File " + f.out + " could not be opened.%N"
        else
          file.start
          file.read_integer
          n := file.last_integer
          file.read_integer
          m := file.last_integer
          if n <= 0 or m <= 0 then
            has_error := True
            error_message := "Maze dimensions not valid.%N"
          else
            from
              i := 0
              create last_maze.make (m, n)
            until
              file.off or has_error or i >= n*m
            loop
              file.read_character
              inspect file.last_character
              when '.' then
                last_maze.set_empty ((i // n) + 1, (i \\ n) + 1)
              when '#' then
                last_maze.set_wall ((i // n) + 1, (i \\ n) + 1)
              when '*' then
                last_maze.set_exit ((i // n) + 1, (i \\ n) + 1)
```

```
                else
48                  if file.last_character.is_space then
                        -- Ignore it
50                      i := i - 1
                    else
52                      has_error := True
                        error_message := "Wrong character " + file.last_character.out + "%N"
54                  end
                end
56              i := i + 1
            end
58          if i < n*m then
                has_error := True
60              error_message := "Maze not filled%N"
            end
62      end
    end
64  end
66
feature -- Access
68
    has_error: BOOLEAN
70      -- Has there been an error when reading?

72  error_message: STRING
        -- Error message
74
    last_maze: MAZE
76      -- Maze

78 end
```

Listing 4: Class *MAZE*

```
class
2   MAZE

4 inherit

6   ARRAY2 [CHARACTER]
        redefine
8           out
        end
10
create
12  make

14 feature -- Constants

16  empty_char: CHARACTER is '.'
        -- Character for empty fields
18
```

```
    exit_char: CHARACTER is '*'
20      -- Character for an exit field

22  wall_char: CHARACTER is '#'
        -- Character for a wall field
24
    visited_char: CHARACTER is 'x'
26      -- Character for a field that has been visited by 'find_path'

28 feature -- Element change

30  set_empty (r, c: INTEGER)
        -- Set field with row 'r' and column 'c' to empty.
32    require
        r_valid: r >= 1 and r <= height
34      c_valid: c >= 1 and c <= width
      do
36      put (empty_char, r, c)
      ensure
38      item (r, c) = empty_char
      end
40
    set_exit (r, c: INTEGER)
42      -- Set field with row 'r' and column 'c' to exit.
      require
44      r_valid: r >= 1 and r <= height
        c_valid: c >= 1 and c <= width
46    do
        put (exit_char, r, c)
48    ensure
        item (r, c) = exit_char
50    end

52  set_wall (r, c: INTEGER)
        -- Set field with row 'r' and column 'c' to wall.
54    require
        r_valid: r >= 1 and r <= height
56      c_valid: c >= 1 and c <= width
      do
58      put (wall_char, r, c)
      ensure
60      item (r, c) = wall_char
      end
62
    set_visited (r, c: INTEGER)
64      -- Set field with row 'r' and column 'c' to visited.
      require
66      r_valid: r >= 1 and r <= height
        c_valid: c >= 1 and c <= width
68    do
        put (visited_char, r, c)
70    ensure
```

```
        item (r, c) = visited_char
    end

feature -- Status report

    is_valid (c: CHARACTER): BOOLEAN
        -- Is 'c' a valid character?
    do
        Result := c = empty_char or c = wall_char or c = exit_char
    end

feature -- Path finding

    path: STRING
        -- Sequence of instructions to find out of the maze

    path_exists: BOOLEAN
        -- Does a path exist?

    find_path (r, c: INTEGER)
        -- Find the path starting at row 'r' and column 'c'.
    do
        if item (r, c) = exit_char then
            path_exists := True
            path := ""
        elseif item (r, c) = empty_char then
            set_visited (r, c)
            if (c-1) > 0 and not path_exists then
                find_path (r, c-1)
                if path_exists then
                    path := "W > " + path
                end
            end
            if (r-1) > 0 and not path_exists then
                find_path (r-1, c)
                if path_exists then
                    path := "N > " + path
                end
            end
            if (c+1) <= width and not path_exists then
                find_path (r, c+1)
                if path_exists then
                    path := "E > " + path
                end
            end
            if (r+1) <= height and not path_exists then
                find_path (r+1, c)
                if path_exists then
                    path := "S > " + path
                end
            end
            set_empty (r, c)
```

```
      end
124   end

126 feature -- Output

128   out: STRING
        -- Output
130   local
        i, j: INTEGER
132   do
        from
134       i := 1
          j := 1
136       Result := ""
        until
138       i > height
        loop
140       from
            j := 1
142       until
            j > width
144       loop
            Result.append_character (item (i, j))
146         j := j + 1
          end
148       i := i + 1
          Result := Result + "%N"
150     end
    end
152
end
```

Listing 5: Class *APPLICATION*

```
1 class
    MAZE_APPLICATION
3
  create
5   make

7 feature -- Initialization

9   make is
        -- Run application.
11    local
        mr: MAZE_READER
13      maze: MAZE
        start_row, start_column: INTEGER
15    do
        create mr
17      io.put_string ("Please enter the name of a maze file: ")
        io.read_line
19      mr.read_maze (io.last_string)
```

```
      if mr.has_error then
21      io.put_string (mr.error_message)
      else
23      maze := mr.last_maze
        io.put_string ("%N" + maze.out + "%N")
25
        io.put_string ("Please enter a starting field for finding a path.%N")
27      from
        until
29        start_row /= 0
        loop
31        io.put_string ("Row: ")
          io.read_integer
33        if io.last_integer > 0 and io.last_integer <= maze.height then
            start_row := io.last_integer
35        else
            io.put_string ("Invalid row. Please try again%N")
37        end
        end
39      from
        until
41        start_column /= 0
        loop
43        io.put_string ("Column: ")
          io.read_integer
45        if io.last_integer > 0 and io.last_integer <= maze.width then
            start_column := io.last_integer
47        else
            io.put_string ("Invalid column. Please try again%N")
49        end
        end
51      maze.find_path (start_row, start_column)
        if maze.path_exists then
53        io.put_string ("There's a way out! Go " + maze.path.out + "You're free!%N
              ")
        else
55        io.put_string ("Oops, no way out! You're trapped!%N")
        end
57    end
    end
59

61 end -- class APPLICATION
```