# AP® COMPUTER SCIENCE AB
# 2007 SCORING GUIDELINES

## Question 4: Environment Iterator (MBS)

| Part A: | next | 5 points |
|---|---|---|

**+1/2**   save current value of `loc`

**+1/2**   correctly access `loc.row()` and `loc.col()`

**+1**   bottom edge case
    **+1/2**   determine if last row
    **+1/2**   `new Location(loc.col()+1, env.numCols()-1);`

**+1**   left edge & non-bottom edge case
    **+1/2**   determine if leftmost column (not on last row)
    **+1/2**   `new Location(0, loc.row()+1);`

**+1/2**   otherwise, `new Location(loc.row()+1, loc.col()-1);`

**+1**   correctly assign `loc` in all three cases

**+1/2**   return saved `loc`


| Part B: | emptyLocs | 4 points |
|---|---|---|

**+1/2**   create list of locations

**+1/2**   `EnvIterator iter = new EnvIterator(env);`

**+2 1/2**   add empty locations to list
    **+1/2**   stop adding if `!iter.hasNext()`
    **+1/2**   stop adding if `n` distinct locations added
    **+1/2**   `iter.next()` in context of loop
    **+1**   add empty location
        **+1/2**   check if location from iterator is empty
        **+1/2**   append empty location to list

**+1/2**   return the list of empty locations

### Question 4: Environment Iterator (MBS)

**PART A:**

```
public Location next()
{
    Location retLoc = loc;
    if (loc.row() == env.numRows()-1) {
        loc = new Location(loc.col()+1 , env.numCols()-1);
    }
    else if (loc.col() == 0) {
        loc = new Location(0, loc.row()+1);
    }
    else {
        loc = new Location(loc.row()+1, loc.col()-1);
    }
    return retLoc;
}
```

**PART B:**

```
public List<Location> emptyLocs(BoundedEnv env, int n)
{
    List<Location> empties = new ArrayList<Location>();

    EnvIterator iter = new EnvIterator(env);
    while (iter.hasNext() && empties.size() < n) {
        Location next = iter.next();
        if (env.isEmpty(next)) {
            empties.add(next);
        }
    }
    return empties;
}
```

Complete method next below.

```
/** Precondition: hasNext() returns true
 *   Postcondition: loc has been updated to the successor location
 *   @return the next location in the environment
 */
public Location next()
{
    Location temp = loc;
    int row = loc.row();
    int col = loc.col();
    int numRows = env.numRows();
    int numCols = env.numCols();
    if(row == numRows-1 && col == numCols -1)
    {
        loc = null;
        return temp;
    }
    if(row == numRows-1)
    {
        loc = new Location(col+1, row);
        return temp;
    }
    if(col == 0)
    {
        loc = new Location(0, row+1);
        return temp;
    }
    loc = new Location(row+1, col-1);
    return temp;
```

Part (b) begins on page 20.

(b) A client class contains the method `emptyLocs`, which returns a list of the first n empty locations when a given square environment `env` is traversed by an `EnvIterator`. If there are fewer than n empty locations in `env`, `emptyLocs` should return all of them.

For example, suppose the environment `env` is as shown in the diagram below where x indicates an occupied location. In this example, the call `emptyLocs(env, 5)` returns a list of locations [(0, 1), (1, 0), (2, 0), (0, 3), (1, 2)].

|   | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | x | 1 | x | 4 |
| 1 | 2 | x | 5 | x |
| 2 | 3 | x |   |   |
| 3 |   | x |   |   |

Complete method `emptyLocs` below.

```
/** @param env the environment over which to iterate
 *           Precondition: env is square, i.e., env.numRows() == env.numCols()
 *    @param n the desired number of empty locations to be returned
 *           Precondition: n > 0
 *    @return a list of the first n empty locations;
 *            all empty locations if there are fewer than n empty locations.
 *            Locations are ordered in the order in which they are visited by the EnvIterator
 */
public List<Location> emptyLocs(BoundedEnv env, int n)
{
    EnvIterator itr = new EnvIterator(env);
    int c = 0;
    ArrayList list = new ArrayList<Location>(n);
    while(itr.hasNext() && c < n)
    {
        Location loc = itr.next();
        if(env.isEmpty(loc))
        {
            c++;
            list.add(loc);
        }
    }
    return list;
```

-20-

Complete method next below.

```
/** Precondition: hasNext() returns true
 *  Postcondition: loc has been updated to the successor location
 *  @return  the next location in the environment
 */
public Location next()
```

if ( loc. row() == rows)            new Location(loc.col()+1, 4);

if ( loc. row() == 0) : loc = new Location (0, loc.row()+1);

else      loc = new Location (loc.row()-1, loc.col()-1);

```
{
  if ( loc.row() == rows){
    loc = new Location (loc.col()+1, 4);
    return loc;
  }
  if ( loc.row() == 0){
    loc = new Location (0, loc.row()+1);
    return loc;
  }
  else{
    loc = new Location (loc.row()-1, loc.col()-1);
    return loc;
  }
}
```

Part (b) begins on page 20.

(b) A client class contains the method `emptyLocs`, which returns a list of the first n empty locations when a given square environment `env` is traversed by an `EnvIterator`. If there are fewer than n empty locations in `env`, `emptyLocs` should return all of them.

For example, suppose the environment `env` is as shown in the diagram below where x indicates an occupied location. In this example, the call `emptyLocs(env, 5)` returns a list of locations [(0, 1), (1, 0), (2, 0), (0, 3), (1, 2)].



Complete method `emptyLocs` below.

```
/** @param env  the environment over which to iterate
 *          Precondition: env is square, i.e., env.numRows() == env.numCols()
 *  @param n  the desired number of empty locations to be returned
 *          Precondition: n > 0
 *  @return a list of the first n empty locations;
 *          all empty locations if there are fewer than n empty locations.
 *          Locations are ordered in the order in which they are visited by the EnvIterator
 */
public List<Location> emptyLocs(BoundedEnv env, int n)
{
    int foundEmpties = 0;
    List<Location> loclist = new ArrayList<Location>();
    Iterator itr = EnvIterator(env);
    while (foundEmpties <= n) {
        while (itr.hasNext()) {
            Location loc = itr.next();

            if (env.isEmpty(loc)) {
                loclist.add(loc);
                foundEmpties ++;
            }
        }
    }

    return loclist;
}
```

**GO ON TO THE NEXT PAGE.**

-20-

Complete method next below.

```
/** Precondition: hasNext() returns true
 *  Postcondition: loc has been updated to the successor location
 *  @return the next location in the environment
 */
public Location next()
```

```
    {
        if (loc.row() == env.numRows()-1 &&
                loc.col() == env.numCols()-1 )
            loc = new Location(
                        env.numRows() - loc.row()
                                loc.row());
        else if (loc.col() == env.numCols()-1)
            loc = new Location(env.numRows() - loc.row(),
                        env.numCols() - loc.
                                col() );
        else
            loc = new Location( loc.row()+1, loc.col()-1);
    }
```

Part (b) begins on page 20.

(b) A client class contains the method `emptyLocs`, which returns a list of the first n empty locations when a given square environment `env` is traversed by an `EnvIterator`. If there are fewer than n empty locations in `env`, `emptyLocs` should return all of them.

For example, suppose the environment `env` is as shown in the diagram below where x indicates an occupied location. In this example, the call `emptyLocs(env, 5)` returns a list of locations [(0, 1), (1, 0), (2, 0), (0, 3), (1, 2)].

|   | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | x |   | x |   |
| 1 |   | x |   | x |
| 2 |   | x |   |   |
| 3 |   | x |   |   |

Complete method `emptyLocs` below.

```
/** @param env  the environment over which to iterate
 *          Precondition: env is square, i.e., env.numRows() == env.numCols()
 *  @param n  the desired number of empty locations to be returned
 *          Precondition: n > 0
 *  @return a list of the first n empty locations;
 *          all empty locations if there are fewer than n empty locations.
 *          Locations are ordered in the order in which they are visited by the EnvIterator
 */
public List<Location> emptyLocs(BoundedEnv env, int n)
```

```
{
    List locations = new List();
    for (int x=0; x<n; x++)
    {
        for (int y=0; y<env.numRows(); y++)
        {
            for (int z=0; z<env.numCols(); z++)
            {   if (env.isEmpty(new Location(y,z)))
                    locations.add(new Location(y,z));
            }
        }
    }
    while (locations.size() > 5)
        locations.remove(locations.size() -1);
    return locations;
}
```

## Question 4

### Overview

This question was based on the Marine Biology Simulation (MBS) case study and focused on implementing and using an iterator on environments. Students were given the framework of an `EnvIterator` class, which contained fields for storing a `BoundedEnv` and the next `Location` to be returned by the iterator. In part (a) they were required to implement the `next` method, following a detailed description of the order to be taken by the iterator. This could be accomplished by either determining the next location's row and column mathematically, or by traversing along a diagonal until the edge of the environment is reached. In part (b) students were required to implement a method from a client class, which uses an `EnvIterator` to traverse an environment, identify empty locations, and collect them in a `List`.

### Sample: AB4a
### Score: 8½

The student received full credit for part (a). The initial check for the lower right corner of the environment is unnecessary, but the student still computes the correct next location and returns the saved location.

In part (b) the student lost the ½ point for creating a list of locations. The statement `ArrayList list = new ArrayList<Location>(n);` fails to include the generic `<Location>` on the left-hand side of the assignment statement.

### Sample: AB4b
### Score: 5½

In part (a) the student lost a ½ point for never saving the original value of `loc`. The student earned a ½ point for correctly accessing `loc.row` and `loc.col` throughout the problem. The student lost both ½ points for the bottom row, since the comparison is to a nonexistent variable `row`, and the new location uses the constant 4 instead of the number of columns in the environment. The student lost the first ½ point for checking the left edge because of a comparison using `loc.row` instead of `loc.col`. The creation of a new location is correct, so the student earned this ½ point. The student lost the ½ point for the otherwise case, since the new location uses `loc.row() - 1` instead of `loc.row() + 1`. The student earned the full point for assigning a new `Location` to `loc` in all three cases. Finally, since there is no saved value of `loc` originally, the student cannot return the saved value and thus lost the return ½ point.

In part (b) the student earned all of the points, except the ½ point designated for stopping when there are `n` empty locations in the list.

**Sample: AB4c**
**Score: 3**

In part (a) the student received the ½ point for correctly accessing `loc.row` and `loc.col` throughout. The student also received the ½ point for creating the correct new location in the otherwise case and 1 full point for assigning a new `Location` to `loc` in all three cases. All other points for part (a) were lost.

In part (b) the student earned the ½ point for adding an empty location to the list and the ½ point for returning a list of empty locations. Because there is no `EnvIterator,` the student did not earn the ½ point for declaring the iterator, stopping when `!iter.hasNext()`, calling `iter.next()`, and checking whether the location returned by the iterator is empty.