AP® COMPUTER SCIENCE A
2009 SCORING GUIDELINES

Question 4: Tile Game

Part (a) `getIndexForFit` **6 points**

+1 empty board
- **+1/2** checks for zero-sized board
- **+1/2** returns 0 if empty board detected

+1 accesses tiles from board
- **+1/2** accesses any tile from board
- **+1/2** accesses all tiles of board (as appropriate) with no out-of-bounds access potential

+1 uses tile values
- **+1/2** accesses left or right value of any tile
- **+1/2** compares left (right) value of parameter with right (left) value of any tile from board

+2 determines tile fit
- **+1/2** only right value of parameter compared with left value of initial tile of board
- **+1/2** only left value of parameter compared with right value of final tile of board
- **+1** compares appropriate values of parameter and interior tiles of board

+1 result
- **+1/2** returns located index if tile fits in board
- **+1/2** returns -1 if tile does not fit in board

Part (b) `insertTile` **3 points**

+1/2 invokes `getIndexForFit` or replicates functionality with no errors

+1 1/2 tile orientation
- **+1/2** invokes `rotate` on parameter
- **+1/2** performs **all** necessary rotations
- **+1/2** invokes `getIndexForFit` for each necessary orientation

+1/2 adds tile correctly and only if `getIndexForFit` returns value other than -1

+1/2 returns `true` if `getIndexForFit` returns value other than -1; `false` otherwise

A4a,

Complete method `getIndexForFit` below.

```
/** Determines where to insert tile, in its current orientation, into game board
 *  @param tile the tile to be placed on the game board
 *  @return the position of tile where tile is to be inserted:
 *              0 if the board is empty;
 *             -1 if tile does not fit in front, at end, or between any existing tiles;
 *             otherwise, 0 ≤ position returned ≤ board.size()
 */
private int getIndexForFit(NumberTile tile)
{
  if(board.get(0).getLeft() == tile.getRight())
     return 0;
  if(board.size() != 1)
  {
    for (int c=0; c<(board.size()-1); c++)
    {
      if (board.get(c).getRight() == tile.getLeft() AND
          board.get(c+1).getLeft() == tile.getRight())
          return c+1;
    }
  }
  if(board.get(board.size()-1).getRight() == tile.getLeft())
      return board.size();
  return -1;
}
```

Part (b) begins on page 18.

A4a2

(b) Write the `TileGame` method `insertTile` that attempts to insert the given tile on the game board. The method returns `true` if the tile is inserted successfully and `false` only if the tile cannot be placed on the board in any orientation.

Assume that `getIndexForFit` works as specified, regardless of what you wrote in part (a).

Complete method `insertTile` below.

```
/** Places tile on the game board if it fits (checking all possible tile orientations if necessary).
 *  If there are no tiles on the game board, the tile is placed at position 0.
 *  The tile should be placed at most 1 time.
 *  Precondition: board is not null
 *  @param tile the tile to be placed on the game board
 *  @return true if tile is placed successfully; false otherwise
 *  Postcondition: the orientations of the other tiles on the board are not changed
 *  Postcondition: the order of the other tiles on the board relative to each other is not changed
 */
public boolean insertTile(NumberTile tile)
{
  for (int c=0; c<4; c++)
  {
    if (getIndexForFit(tile) != -1)
    {
      board.add (getIndexForFit(tile), tile);
      return true;
    }
    else
      {tile.rotate();}
  }
  return false;
}
```

GO ON TO THE NEXT PAGE.

A4b,

Complete method `getIndexForFit` below.

```
/** Determines where to insert tile, in its current orientation, into game board
 *  @param tile the tile to be placed on the game board
 *  @return the position of tile where tile is to be inserted:
 *            0 if the board is empty;
 *           -1 if tile does not fit in front, at end, or between any existing tiles;
 *           otherwise, 0 ≤ position returned ≤ board.size()
 */
private int getIndexForFit(NumberTile tile)
{ INT  LEFT = TILE.GETLEFT();
  INT  RIGHT = TILE.GETRIGHT();

FOR(INT I=0; I <= BOARD.SIZE(); I++)
    { INT  TEMPLEFT = BOARD.GET(I).GETLEFT();
      INT  TEMPRIGHT = BOARD.GET(I).GETRIGHT();
     IF(RIGHT == TEMPLEFT && I==0)
        { BOARD.ADD(I, TILE);
          RETURN I;
        }
     IF(LEFT == TEMPRIGHT && I== BOARD.SIZE()-1)
        {BOARD.ADD(TILE);
          RETURN I+1;
        }
     IF(LEFT == TEMPRIGHT)
        {IF(RIGHT== BOARD.GET(I+1).GETLEFT())
            {BOARD.ADD(I+1,TILE);
             RETURN I+1;
            }
        }
    }
RETURN -1;
}
```

Part (b) begins on page 18.

GO ON TO THE NEXT PAGE.

A4b2

(b) Write the `TileGame` method `insertTile` that attempts to insert the given tile on the game board. The method returns `true` if the tile is inserted successfully and `false` only if the tile cannot be placed on the board in any orientation.

Assume that `getIndexForFit` works as specified, regardless of what you wrote in part (a).

Complete method `insertTile` below.

```
/** Places tile on the game board if it fits (checking all possible tile orientations if necessary).
 *  If there are no tiles on the game board, the tile is placed at position 0.
 *  The tile should be placed at most 1 time.
 *  Precondition: board is not null
 *  @param tile the tile to be placed on the game board
 *  @return true if tile is placed successfully; false otherwise
 *  Postcondition: the orientations of the other tiles on the board are not changed
 *  Postcondition: the order of the other tiles on the board relative to each other is not changed
 */
public boolean insertTile(NumberTile tile)
{ INT INDX = 0;

  FOR(INT I = 0; I < 4; I++)
    { INDX = GETINDEXFORFIT(TILE);
      IF (INDX == -1)
        { TILE.ROTATE(); }
      ELSE
        RETURN TRUE;
    }

  RETURN FALSE;

}
```

GO ON TO THE NEXT PAGE.

A4c, 1

Complete method `getIndexForFit` below.

```
/** Determines where to insert tile, in its current orientation, into game board
 *  @param tile the tile to be placed on the game board
 *  @return the position of tile where tile is to be inserted:
 *            0 if the board is empty;
 *           -1 if tile does not fit in front, at end, or between any existing tiles;
 *           otherwise, 0 ≤ position returned ≤ board.size()
 */
private int getIndexForFit(NumberTile tile)
{
   int tileLeft = tile.getLeft();
   int tileRight = tile.getRight();
   int bestPosition = 0;

   if (board.size = 0)
   {
     return (0)
   }

   if (board[0] = tileRight)
   {
     return (0)
   }
   if (board[board.size -1] = tileLeft)
   {
     return (board.size)
   }
   for (int i=0; i< board.size -1; i++)
   {
     if (board[i].getRight = tileLeft && board[i+1].getLeft = tileRight)
     {
       bestPosition = i
     }
   }
   return (bestPosition)
}
```

Part (b) begins on page 18.

A4c2

(b) Write the `TileGame` method `insertTile` that attempts to insert the given tile on the game board. The method returns `true` if the tile is inserted successfully and `false` only if the tile cannot be placed on the board in any orientation.

Assume that `getIndexForFit` works as specified, regardless of what you wrote in part (a).

Complete method `insertTile` below.

```
/** Places tile on the game board if it fits (checking all possible tile orientations if necessary).
 *  If there are no tiles on the game board, the tile is placed at position 0.
 *  The tile should be placed at most 1 time.
 *  Precondition: board is not null
 *  @param tile the tile to be placed on the game board
 *  @return true if tile is placed successfully; false otherwise
 *  Postcondition: the orientations of the other tiles on the board are not changed
 *  Postcondition: the order of the other tiles on the board relative to each other is not changed
 */
public boolean insertTile(NumberTile tile)
{
   int bestPosition = getIndexForFit;


   board.add(tile, bestPosition);
   return (true)
}
```

AP® COMPUTER SCIENCE A
2009 SCORING COMMENTARY

Question 4

Overview

This question focused on object-oriented programming, algorithm development, and list processing, requiring significant problem analysis and algorithm design. Students were expected to analyze the problem, design algorithms, and then implement them using the well-specified public interface of a `NumberTile` class. The question involved writing two related methods for a given `TileGame` class. The method `getIndexForFit` determines where a given `NumberTile` object can be added into the `ArrayList` instance variable `board`. The method `insertTile` modifies the state of a `TileGame` object by adding a given `NumberTile` object to `board`. In designing their solutions, students were required to consider the full set of potential cases and to work within the constraints of the public methods provided for `NumberTile`. This question assessed the ability to decompose a stated problem into computational constituents, use algorithmic thinking, rely on abstraction, and demonstrate facility with an `ArrayList` object.

Sample: A4a
Score: 8

The solution presented for part (a) lost the first two ½ points for not checking to see if the board is empty. The solution earned the rest of the points in this section by accessing tiles from the board and properly comparing them with the parameter `tile` as it exists in its current orientation. The solutions checks to see if the tile can be placed at the beginning, middle, and end of the list. The solution correctly returns the index of where the tile is to be placed in the board, including a return of 0 if the board is empty, or –1 if the tile cannot be placed in the board. Part (a) earned 5 points.

The solution presented for part (b) earned all available points for this section. A proper call to the method `getIndexForFit(tile)` is executed in conjunction with calls to `tile.rotate()`. A loop is used to determine placement of the parameter `tile` on the board. The tile is added to the board and the method returns `true` if `getIndexForFit(tile) != -1`; otherwise the method returns `false`.

Sample: A4b
Score: 6

The solution presented for part (a) lost the first two ½ points for failing to check if the board is empty. The solution earned the next ½ point for properly obtaining a tile from the board but lost the next ½ point by attempting to access values beyond `board.size()-1.` The solution earned the rest of the points in this section by making the proper comparisons of the parameter `tile` as it exists in its current orientation. The solution checks to see if the tile can be placed at the beginning, middle, and end of the list. The solution correctly returns the index of where the tile is to be placed in the board, including a return of 0 if the board is empty or –1 if the tile cannot be placed in the board. Part (a) earned 4½ points; however, 1 point was deducted under the general usage guidelines because the code actually inserts the tile into the board, causing an unspecified side effect that changes the state of the data structure.

The solution presented for part (b) earned 2½ points. A proper call to the method `getIndexForFit(tile)` is executed in conjunction with calls to `tile.rotate()`. A loop is used to determine placement of the parameter `tile` on the board. The tile never actually gets added to the board, so ½ point was lost. The solution returns the appropriate value based on whether or not a placement was found for the tile.

Question 4 (continued)

Sample: A4c
Score: 3.5 (rounded to 4)

The solution presented for part (a) earned the first two ½ points for determining if the board is empty. The solution lost the next ½ point for not accessing a tile from the board properly but earned the ½ point for accessing all tiles of the board and the next two ½ points for using the tile values. The solution lost two ½ points for failing to check if the parameter `tile` can be added at either end of the board. The solution earned 1 point for searching the interior tiles for placement consideration. The last two ½ points were not earned due to improper `return` statements. Part (a) earned 3½ points.

The solution presented for part (b) earned no points. The solution has an improper call to `getIndexForFit(tile)` and reverses the order of the parameters of the call to `board.add()`.