Iverson computing competition
2016 may 31

name _____

school _____

city _____

grade _____

cs teacher _____

are you taking AP computer science? (yes/no) ____
are you taking IB computer science? (yes/no) ____
have you taken advanced level courses? (3000 level, e.g. CSE3110 iterative algorithms I.) (yes/no/currently taking) _____

illegible answers will not be marked

question	- - - -	marks	your score
1	tiling	11	
2	ascii maze	12	
3	blorks	10	
4	hex	7	
total	- - - -	40	

Exam prepared by Zachary Friggstad and Ryan Hayward

Exam Format

This is a two-hour paper and pencil exam. There are four questions, each with multiple parts. Some part(s) might be easy. Solve as many parts of as many questions as you can.

Programming Language

Questions that require programming can be answered using any language (e.g. C/C++, Java, Python, ...) or pseudo-code. **Pseudo-code should be detailed enough to allow for a near direct translation into a programming language.** Clarify your code with appropriate comments. For full marks, an answer must be correct, well-explained, and as simple as possible.

Our primary interest is in thinking skill rather than coding wizardry, so logical thinking and systematic problem solving count for more than programming language knowledge.

Suggestions

1. You can assume that the user enters only valid input in the coding questions.

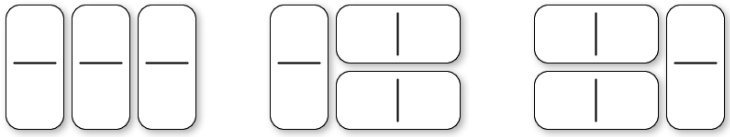
2. In somes cases, sample executions of the desired program are shown. Review the samples carefully to make sure you understand the specifications. The samples may give hints.

3. Design your algorithm before writing any code. Use any format (pseudo-code, diagrams, tables) or aid to assist your design plan. We may give part marks for legible rough work, especially if your final answer is lacking. We are looking for key computing ideas, not specific coding details, so you can invent your own "built-in" functions for simple subtasks such as reading the next number, or the next character in a string, or loading an array. Make sure to specify such functions by giving a relationship between their inputs and outputs.

4. Read all questions before deciding which ones to attempt, and in which order. Start with the easiest parts of each question.

question 1: tiling

We want to tile $2 \times n$ grids using dominoes. A domino is a 2×1 or 1×2 tile. Here are all ways to tile a 2×3 grid:



(a) [1 mark] Draw all ways to tile a 2×4 grid.

(b) [2 marks] Let $f(n)$ denote the number of ways to tile a $2 \times n$ grid. So $f(0) = 1, f(1) = 1, f(2) = 2$, and $f(3) = 3$. Find $f(5)$ and $f(6)$ (you do not have to draw any tilings, but you can if it helps you).

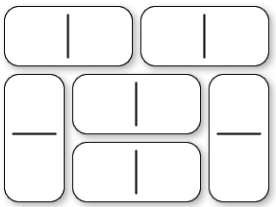
$f(5) =$

$f(6) =$

(c) [2 marks] Let $n \geq 2$. Assume that we know $f(k)$ for all values $0 \leq k \leq n-1$. Give a simple expression that calculates $f(n)$ using these known $f(k)$ values. Carefully justify your answer using at most **two sentences**.

(d) [2 marks] Write a function `tiling(n)` that returns the value $f(n)$ for a given integer $n \geq 0$. For full marks, it should compute $f(9999)$ in less than one second.

(e) [2 mark] Let $g(n)$ denote the number of ways to tile a $3 \times n$ grid using dominoes. Here is a tiling of a 3×4 grid:



So, $g(0) = 1, g(2) = 3$, $g(4) = 11$, and $g(6) = 41$.

What is $g(n)$ when n is odd? Justify your answer.

(f) [2 marks] Let $n \geq 2$ be an even integer. Assume that we know $g(k)$ for all $0 \leq k \leq n - 1$. Give an expression that easily calculates $g(n)$ using these known $g(k)$ values. Justify your answer.

question 2: ascii mazes

One way to represent a maze using ASCII characters is |-*format*, which uses vertical bars |, underscores _, and spaces. A rectangular grid using only these three characters is a *valid maze* if

- there are at least 2 rows and at least 3 columns, and
- the top row has only _ characters, and
- the first character in the second row is a space, indicating the entrance, and
- the last character in the last row is _, indicating the exit, and
- except for the entrance and exit, the first and last character on each row is |, and
- the bottom row has no spaces.

Example i

```
 _________
|  ___  |
| | | | |
|_____|__
```

In |-format, a grid cell is *vacant* if it is a space, or if it is not on the top row and is _. Two vacant cells *join each other* if one is beside the other (up, down, left, or right). Neighbouring vacant cells join each other if and only if no wall separates them.

(a) [1 marks] For this maze, put a dot in each vacant cell, and draw a line between each pair of vacant cells that join each other.

```
 ____
   _|
|  _|
|___
```

Another way to represent a maze is with X-*format*. X-format uses a space for vacant cells and an X otherwise. The first character of the second row is a space, for the entrance; the last character of the second-last row is a space, for the exit; every other character on the rectangular boundary of this grid is X.

Example ii

```
XXXXXXXXX
  X     X
X X XXX X
X X X X X
X X X X X
X     X
XXXXXXXXX
```

We can convert from |-format to X-format by adding extra rows, and placing a space or X between two cells that are on top of each other indicating whether they join each other. The X-format example above is what we get by converting from the |-format example. Notice that the number of vacant cells can change during this conversion. The maze in (b) is what you would get by converting from the maze in (a).

(b) [1 marks] For this maze, put a dot in each vacant cell, and draw a line between each pair of vacant cells that join each other.

```
XXXX
   X
X XX
X  X
X XX
X
XXXX
```

(c) [4 marks] Write a function `convert(maze)` that takes an array or list of strings representing a valid maze in `|`-format and prints the corresponding maze in `X`-format. For example, `maze` is from (a) then `convert(maze)` prints the maze from (b).

(d) [6 marks] Now write a function `search(maze)` that takes an array of strings representing a maze in `X`-format. It should print a path from the entrance to the exit using `*` characters. You may assume that there is exactly one way to travel from the entrance to the exit using a path that does not visit a cell more than once.

Example: calling `search(maze)` with `X`-format example ii gives this output:

```
XXXXXXXXX
**X*****X
X*X*XXX*X
X*X*X X*X
X*X*X X*X
X***  X**
XXXXXXXXX
```

question 3: blorks

A *binary string* contains only `0`s and `1`. A *blork* is a string containing only characters `0`, `1`, and `*`. A binary string `str` *matches* a blork `blrk` if each `*` character in `blrk` can be replaced with a binary string (possibly empty, and the binary strings do not all have to be the same) so that the resulting string equals `str`. We show such a matching by starting with the blork and then replacing each `*` with `(b)` where `b` is the needed binary string for that `*`.

Example

- `str = "01001"`, `blrk = "01*01"`, **match** `"01(0)01"`
- `str = "101"`, `blrk = "10**1"`, **match** `"10()()1"`
- `str = "10101101"`, `blrk = "10*10*"`, **match** `"10(101)10(1)" and "10()10(1101)"`
- `str = ""`, `blrk = ""` (both strings are empty), **match** `""`
- `str = "11101"`, `blrk = "101*"`, **no match possible**

(a) [2 marks] For each of the following, indicate whether it matches. If yes, give one replacement, in the same form as the example.

- `str = "1001"`, `blrk = "*1*0*1*"`

- `str = "1101100010101001"`, `blrk = "1101*100*10*01*"`

- `str = "10001101001"`, `blrk = "100*11*101"`

- `str = ""`, `blrk = "*"`

- `str = "10010101010000101000101110110 1"`, `blrk = "*1001*001*11*101"`

- `str = "000110101101100010101010101101"`, `blrk = "*1101*101010*110"`

(b) [3 marks] Write a function `extract(blrk)` that takes a blork `blrk` as parameter and returns an array or list of strings with the binary subtring *pieces* of `blrk`, namely the nonempty binary substrings left over if each $*$ is replaced with a space.

Example: for `blrk = "*10**1110*110"` the three pieces are `"10", "1110", "110"`.

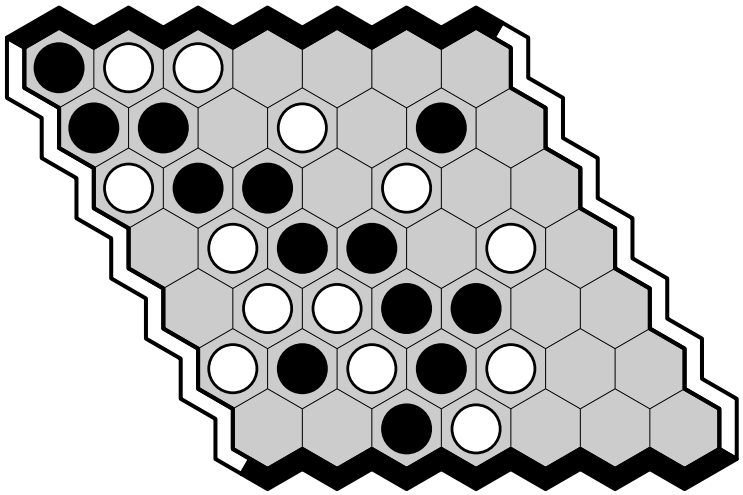
(c) [5 marks] Write a function `match(str, blrk)` that takes a binary string `str` and blork `blrk` as parameters and returns `true` if `str` matches `blrk` and `false` otherwise. If it is helpful, you may use the function described in part (b) even if you did not answer that question.

question 4: hex

Warning: this question can take time. Budget your time wisely.

The two-player game of Hex is played on an $n \times n$ board with hexagonal cells. Below is a 7×7 board. Players alternate turns. On a turn, a player puts a stone on an empty cell.

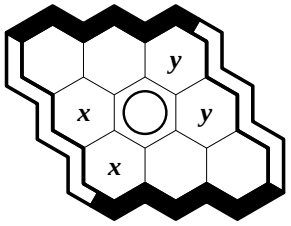
The first player is called *white* and places white stones. The second player is called *black* and places black stones. Two opposing sides of the board are white and the other two opposing sides are black. The winner is whoever connects their two sides with a connected path of their stones. In the example below, black has won.



An amazing property of Hex is that if the board is completely covered with stones then exactly one player has joined their two sides. So draws are not possible.

This question is about playing perfectly in Hex. We say that a player plays *perfectly* if, on each move, if there is some move that is part of a winning strategy for that player, then the player makes such a move. So, whoever can win always makes a winning move; whoever cannot win can play anywhere.

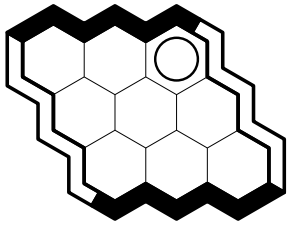
(a) [3 marks] On this 3×3 board white played her first move in the middle. (The letters are for the analysis below.) Now it is black's turn. We claim that white can win.



White can guarantee she gets at least one of the cells labelled x by placing a stone on one of these two cells if black ever places a stone on the other. Similarly, white can guarantee she gets at least one of the cells labelled y. But each x cell touches the left side and the middle, and each y cell touches the middle and the right side. So white will join her two sides if she follows this strategy.

Now assume that white starts a game by by placing her first stone as shown below. It is black's turn. Who will win the game if both players play perfectly from now on? Justify your answer by giving the winning player's strategy **as concisely as possible**.

(We have included a page with blank 3×3 hex grids on the second last page of this exam. You can use these to help describe the strategy, just make sure that we can follow your reasoning.)

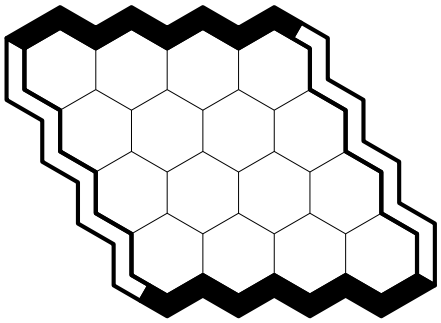


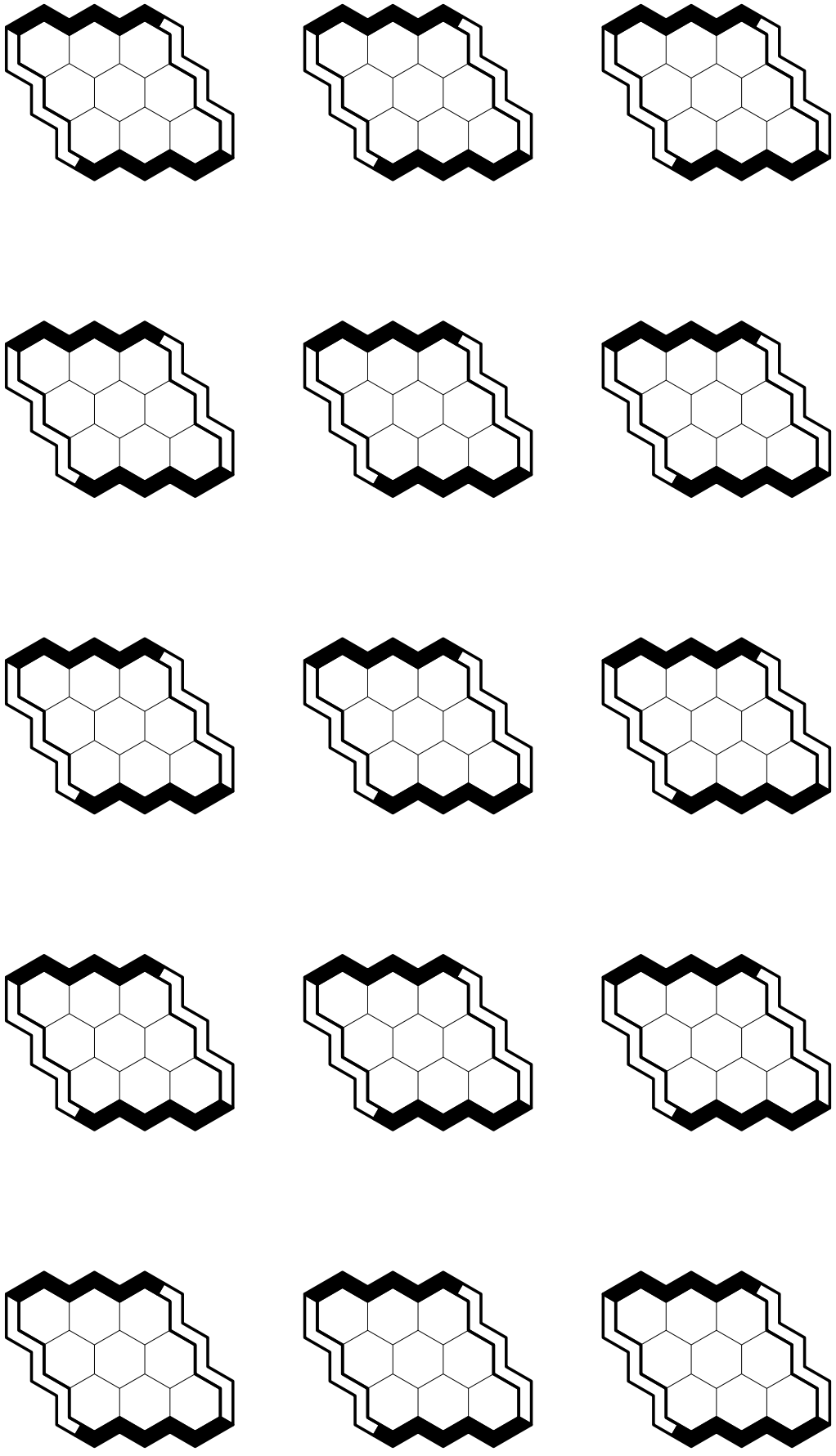
(b) [4 marks] For **each** cell on the 4×4 board below, show who wins (white or black) if that cell is white's first move, and both players play perfectly from then on: if white wins, then on that cell draw an empty circle (or the letter `W`); if black wins, then on that cell draw a filled circle (or the letter `B`).

You do not need to justify your answer: your score will be calculated as follows:

$$\max\left\{\frac{(\#\text{ of correct answers}) - (\#\text{ of incorrect answers})}{4},\quad 0\right\}.$$

Cells left blank will be counted neither correct nor incorrect. Illegible entries will be counted incorrect. (Some empty 4×4 grids can be found at the end of this exam. These are for your scratch notes only and will not be looked at when we grade.)





this page will not be graded