# Assignment 3: Domineering

Due: 20:00, Sat 23 Oct 2021 File name: `domineering.cpp` Full marks: 100

## Introduction

The objective of this assignment is to practice (1) defining functions (being a callee), (2) calling functions (being a caller), and (3) representing special kind of data.

You will implement a two-player paper-and-pencil game called *Domineering*. Two players take turns to put their dominoes to a $4 \times 4$ grid. Each domino occupies two consecutive spaces in the grid. Player 1 must place his/her dominoes vertically, while Player 2 must place his/her dominoes horizontally. The player who can no longer put a domino in his/her turn is the loser, and the other player is the winner. This game can never result in a draw: even when the grid is full, the last player who puts a domino is considered the winner (because the next player has no places to put). Figure 1 shows an example series of game play. The dominoes put by Players 1 and 2 are marked as A and B respectively.

```
      . A B B    Player 1     A A B B    Player 2       A A B B
… →   . A . .       →         A A . .       →           A A . .    Player 2 wins.
      A . B B   (vertical     A . B B   (horizontal     A . B B
      A . . .   domino A)     A . . .    domino B)      A B B .
```

Figure 1: An Example Domineering Game Play

## Program Specification

This section describes the representation of a grid in a game, the necessary functions, and the flow of your program.

### Basic Requirements

- You *cannot declare any global variables* (variables declared outside any functions).
- You *cannot use any functions in the `<cmath>` library*.
- You *cannot use any arrays or vectors.*
- You *cannot use the `string` class*.

### Grid Representation

There are 16 possible positions in a grid. Therefore, we use integers 1 to 16 to denote these positions, where 1–4 denote the spaces in the top row of a grid left-to-right, 5–8 denote the spaces in the second row, 9–12 denote the spaces in the third row, and 13–16 denote the spaces in the bottom row. The positions are basically ordered top-to-bottom, left-to-right, as illustrated in Figure 2.

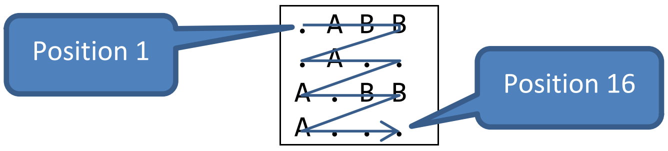

Figure 2: Grid Space Positions

To encode the whole grid of a game, we use a 16-digit integer $d_1d_2d_3d_4d_5d_6d_7d_8d_9d_{10}d_{11}d_{12}d_{13}d_{14}d_{15}d_{16}$. Each digit $d_i$ stores the *state* of position $i$, which has only three possible values: 0, 1, or 2. Value 0 denotes an empty (unfilled) space in the position. Value 1 denotes that the space is filled with a symbol A (by Player 1). Value 2 denotes that the space is filled with a symbol B (by Player 2). E.g., the middle grid in Figure 1 can be encoded by the integer 1122110010221000. Using this representation, an empty grid (with no spaces filled) is simply encoded as 0. (Note: in C++, integer constants should *NOT contain leading zeroes*, otherwise, they will be treated as octal numbers. Thus, the grid in Figure 2, e.g., is 122010010221000 rather than 0122010010221000.)

The data type `int` in C++ is typically 32-bit and thus not big enough to store a 16-digit integer. In order to store a game grid, you shall use a variable of a bigger integer type called `long long`. In Visual Studio and Xcode, `long long` is a 64-bit signed integer type, whose range is $-9{,}223{,}372{,}036{,}854{,}775{,}808 \ldots 9{,}223{,}372{,}036{,}854{,}775{,}807$.

## Required Functions

Your program must contain the following functions, and they must be called *somewhere* in your program. You must *not* modify the prototypes of all these functions. You can design extra functions if you find necessary.

In the functions below, you can assume that (a) parameter `grid` is a proper encoding of the grid of a Domineering game (that is, 16-digit integer containing digits 0/1/2 only) and (b) parameter `p` is either 1 or 2. There can be more assumptions in each individual function as stated in the description. Besides, *all the functions below will be graded individually.* That is, we shall replace your `main()` with our testing code to call the functions one by one for grading. So your code for each function shall implement the description of that function only. You shall *not* write any code in a function that is beyond that function's description.

### `int gridState(long long grid, int pos)`

This function returns the state of position `pos` of `grid`, that is, 0 for empty, 1 for symbol A, and 2 for symbol B. E.g., the call `gridState(1122110010221000, 5)` shall return 1, while the calls `gridState(1220100102210, 12)` and `gridState(1220100102210, 2)` shall return 0. (Note that positions 1 to 3 are empty in 1220100102210.) You can assume that parameter `pos` is always 1–16 in this function. Calling this function will be useful when implementing some of the required functions below.

### `void printGrid(long long grid)`

Prints `grid` to the screen using the format in Figure 1.

### `bool isPlaceable(long long grid, int pos, int p)`

Returns true if Player `p` can put a domino to position `pos` of `grid`. Returns false otherwise. Note the following requirements of this function:

- If `pos` is not 1–16, the function shall return false.
- If `p` is neither 1 nor 2, the function shall return false.
- The domino is not really put to `grid`. The function just checks whether the domino is "placeable" to the grid or not.

- A domino occupies two spaces in a grid. When p is 1 (Player 1), you have to check whether position pos and the cell immediately below (vertical) are empty or not. When p is 2 (Player 2), you have to check whether position pos and the cell immediate to its right (horizontal) are empty or not. Note that Player 1 (*resp.* 2) cannot put a domino when pos in the bottom row (*resp.* rightmost column).

### `void putToGrid(long long &grid, int pos, int p)`

This function performs the task of player p putting a domino (occupying two spaces) to position pos of the reference parameter grid. That means for Player 1, the states of position pos and the position immediately *below* shall be updated to 1. For Player 2, the states of position pos and the position immediately *to its right* shall be updated to 2. Your task is to update the value of the reference parameter grid correctly. The following shows some sample function calls and the expected results.

| Parameter grid | Parameter pos | Parameter p | Value of grid *after* calling putToGrid(grid, pos, p) |
|---:|---:|---:|---:|
| 1122110010221000 | 14 | 2 | 1122110010221220 |
| 122010010221000 | 1 | 1 | 1122110010221000 |
| 0 | 6 | 1 | 10001000000 |
| 122010010011221 | 10 | 2 | 122010012211221 |

Note the following requirements of this function:

- You can assume that parameter pos is always 1–16 in this function.
- You can assume in this function that the original states of the relevant positions must be 0. That is, the spaces are empty for putting the domino successfully. (You check validity *outside* of this function. See steps 3–4 of the next section.)
- Do *not* print anything in this function.

## Program Flow

The program flow of the game is described as follows. You should call the functions above to aid your implementation.

1. The program starts the game with an empty grid (0). Player 1 takes the first turn.
2. Then, prompt the current player to enter a position to put a domino in. Users will *always enter integers* here. For Player 1 (*resp.* 2), the input position means the domino occupies the input space and its immediate below (*resp.* right) position. (E.g., Player 1 entering $\boxed{5}$ means putting a domino to positions 5 and 9.)
3. A domino cannot be placed ("is not placeable") to the input position if the position is not 1–16 or the two relevant spaces (vertical or horizontal) are not empty. In case it is not placeable, display a warning message and go back to Step 2.
4. Update the grid by putting the domino to the corresponding positions.
5. Swap player to take the next turn.
6. Repeat steps 2–5 until a player has no more placeable positions for a domino.
7. When a game finishes, display the message *"Player 1 wins!"* or *"Player 2 wins!"* accordingly. (Recall that there can never be a draw game.)

## Sample Run

The following shows a sample run. The blue text is user input and the other text is the program printout. You can try the provided sample program for other input. *Your program output should be exactly the same as the sample program* (same text, symbols, letter case, spacings, etc.). Note that there is a space after ':' in the printout.

```
. . . .
. . . .
. . . .
. . . .
Player 1's move: 5↵
. . . .
A . . .
A . . .
. . . .
Player 2's move: 0↵
Invalid! Try again.
Player 2's move: 20↵
Invalid! Try again.
Player 2's move: 4↵
Invalid! Try again.
Player 2's move: 9↵
Invalid! Try again.
Player 2's move: 3↵
. . B B
A . . .
A . . .
. . . .
Player 1's move: 14↵
Invalid! Try again.
Player 1's move: 11↵
. . B B
A . . .
A . A .
. . A .
Player 2's move: 13↵
. . B B
A . . .
A . A .
B B A .
Player 1's move: 2↵
. A B B
A A . .
A . A .
B B A .
Player 2's move: 7↵
. A B B
A A B B
A . A .
B B A .
Player 1's move: 12↵
```

```
. A B B
A A B B
A . A A
B B A A
Player 1 wins!
```

## Submission and Marking

- Your program file name should be domineering.cpp. Submit the file in Blackboard (https://blackboard.cuhk.edu.hk/).
- Insert *your name*, *student ID*, and *e-mail* as comments at the beginning of your source file.
- You can submit your assignment multiple times. Only the latest submission counts.
- Your program should be *free of compilation errors and warnings*.
- Your program should *include suitable comments as documentation*.
- ***Do NOT plagiarize.*** Sending your work to others is subjected to the same penalty as the copier.