CSE141 Introduction to Programming (Fall'23)

Lab Examination

Max Marks: 0 *Time Allowed:* 1 ½ hours

Exercise 1 ....

Stock Picker. Create a function that takes an array of integers that represent the amount in dollars that a single stock is worth, and return the maximum profit that could have been made by buying stock on day x and selling stock on day y where $y > x$. Your function should be declared as follows:

```
int stock_picker(int* stocks, int n)
```

where `stocks` is the array of stock prices and `n` is the size of the array.

If the given array is: `{44, 30, 24, 32, 35, 30, 40, 38, 15}`, your program should return `16` because at index 2 the stock was worth $24 and at the index 6 the stock was then worth $40, so if you bought the stock at 24 and sold it on 40, you would have made a profit of $16, which is the maximum profit that could have been made with this list of stock prices.

If there is no profit that could have been made with the stock prices, then your program should return `-1` (e.g. `{10, 9, 8, 2}` should return `-1`).

Submission: `stock.cpp`

Exercise 2 ....

Persistent Little Bugger. Create a recursive function that takes a positive integer and returns its *multiplicative persistence*, which is the number of times you must multiply the digits in the number until you reach a single digit. Your function should not use any loops. You may create helper function if needed.

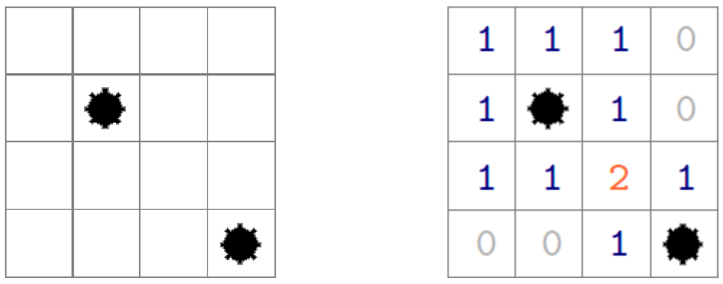
Examples:

- `bugger(39)` → `3`
  Since $3 \times 9 = 27$, $2 \times 7 = 14$, $1 \times 4 = 4$, and 4 has only one digit.
- `bugger(4)` → `0`
  Since 4 is already a one-digit number.

Submission: `bugger.cpp`

Exercise 3 ....

Reverse Minesweeper. Minesweeper is a popular game requiring skill and a little bit of luck. If you're not familiar with it, it involves a minefield with rows and columns. Each position of the field can have a mine or not. For example, the minefield in the figure below (left) has two mines:



However, the player cannot see where the mines are. Instead, she must select a position of the minefield; if the position has a mine, the game is over. If not, a number is revealed. This number indicates how many mines are present in the positions immediately adjacent to that position (including the diagonal ones). For example, the above board would contain the numbers shown in the figure above (right).

Write a C++ function `mines_count()` that takes in a 2D boolean array of size $m \times n$ and returns a 2D integer array of the same size where each cell contains the number of mines in its neighborhood. If the cell itself contains a mine, it should contain the integer 99. A cell's neighborhood consists of the 8 cells surrounding it (including diagonals). The function should have the following signature:

```
int** mines_count(bool** mines, int m, int n)
```

For example, the 2D boolean array shown below (left) represents a 4×4 grid with 2 mines (represented by **`true`**) and 14 empty cells (represented by **`false`**). The function should return the 2D integer array shown on the right.

```
{ {false, false, false, false},
  {false, true,  false, false},
  {false, false, false, false},
  {false, false, false,  true} }
```

```
{ {1,  1,  1,  0},
  {1, 99,  1,  0},
  {1,  1,  2,  1},
  {0,  0,  1, 99} }
```

Submission: `minesweeper.cpp`

Test your function to make sure it works. Here is one test function:

```
int main() {
  const int m = 4, n = 4;
  bool* grid[m];
  grid[0] = new bool[n] {false, false, false, false};
  grid[1] = new bool[n] {false, true, false, false};
  grid[2] = new bool[n] {false, false, false, false};
  grid[3] = new bool[n] {false, false, false, true};

  // call mines_count()
  int** counts = mines_count(grid, m, n);

  // print counts
  for (int i = 0; i < m; ++i) {
      for (int j = 0; j < n; j++)
          std::cout << counts[i][j] << " ";
      std::cout << "\n";
  }

  // free memory
  for (int i = 0; i < m; ++i)
      delete[] grid[i];
}
```