C++ Programming

Multidimensional Arrays

Practice 2

Mostafa S. Ibrahim

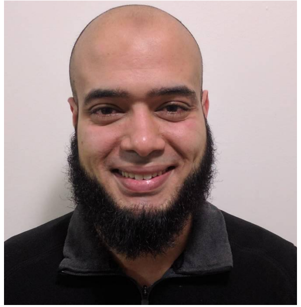
Teaching, Training and Coaching since more than a decade!

Artificial Intelligence & Computer Vision Researcher

PhD from Simon Fraser University - Canada

Bachelor / Msc from Cairo University - Egypt

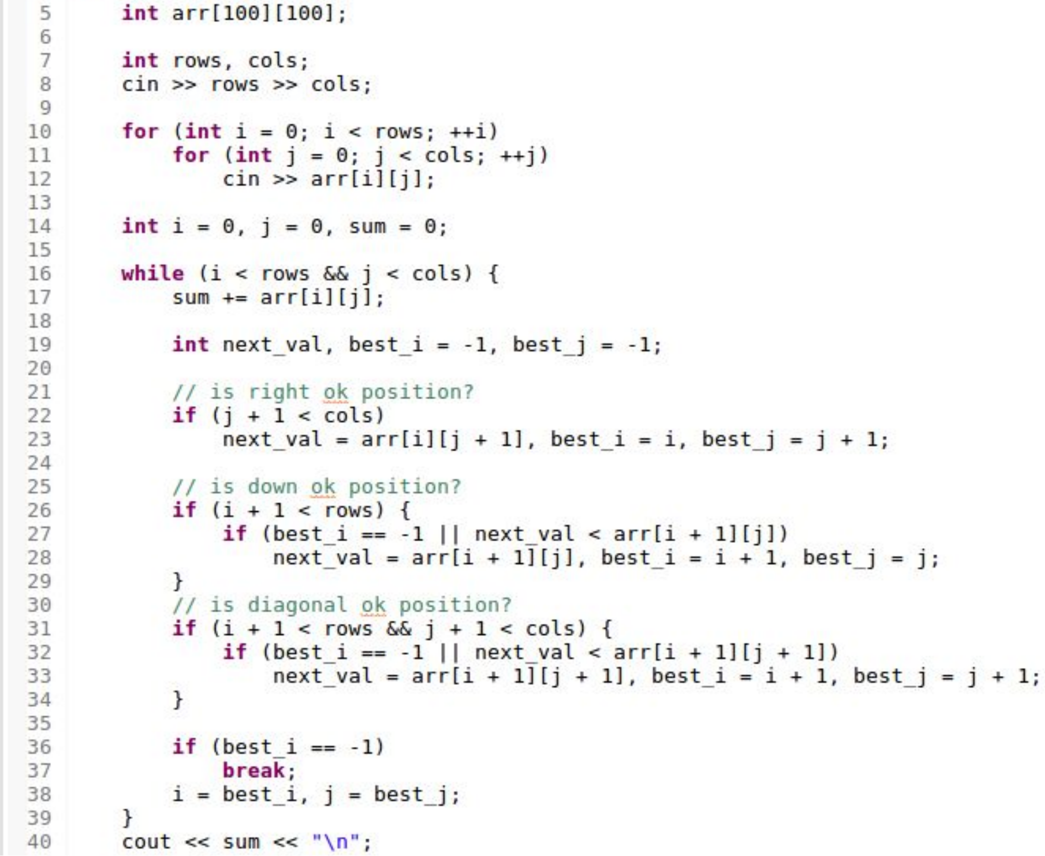
Ex-(Software Engineer / ICPC World Finalist)



Practice: Greedy Robot

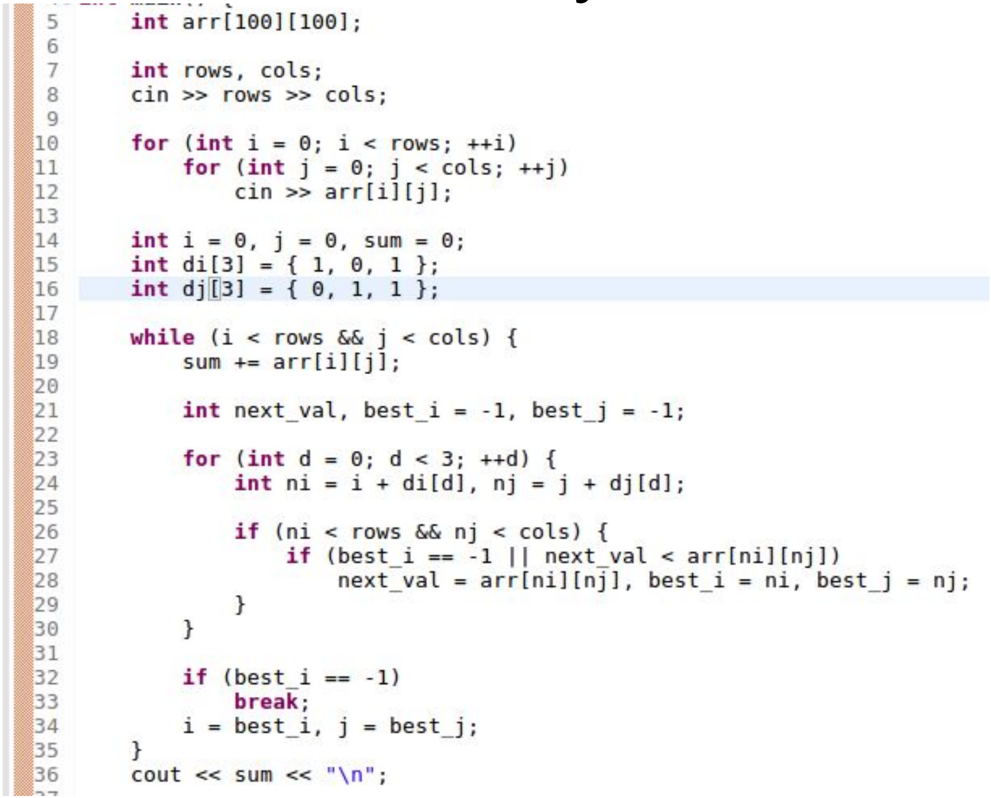
- Read integers N, M, then Read **matrix** NxM. All values are *distinct*.
- A robot starts at cell (0, 0).
- Take the value in the current cell and moves.
- It can move only one step to either: *Right, Bottom or the diagonal.*
- It always selects the cell that has maximum value.
- Print the total values the robot collects

Practice: Greedy Robot



```
    int arr[100][100];

    int rows, cols;
    cin >> rows >> cols;

    for (int i = 0; i < rows; ++i)
        for (int j = 0; j < cols; ++j)
            cin >> arr[i][j];

    int i = 0, j = 0, sum = 0;

    while (i < rows && j < cols) {
        sum += arr[i][j];

        int next_val, best_i = -1, best_j = -1;

        // is right ok position?
        if (j + 1 < cols)
            next_val = arr[i][j + 1], best_i = i, best_j = j + 1;

        // is down ok position?
        if (i + 1 < rows) {
            if (best_i == -1 || next_val < arr[i + 1][j])
                next_val = arr[i + 1][j], best_i = i + 1, best_j = j;
        }
        // is diagonal ok position?
        if (i + 1 < rows && j + 1 < cols) {
            if (best_i == -1 || next_val < arr[i + 1][j + 1])
                next_val = arr[i + 1][j + 1], best_i = i + 1, best_j = j + 1;
        }

        if (best_i == -1)
            break;
        i = best_i, j = best_j;
    }
    cout << sum << "\n";
```

Practice: Greedy Robot - Shorter

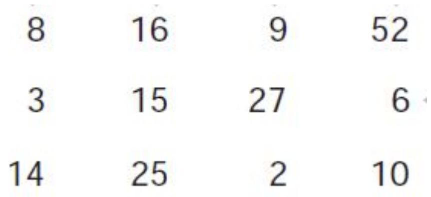


```
    int arr[100][100];

    int rows, cols;
    cin >> rows >> cols;

    for (int i = 0; i < rows; ++i)
        for (int j = 0; j < cols; ++j)
            cin >> arr[i][j];

    int i = 0, j = 0, sum = 0;
    int di[3] = { 1, 0, 1 };
    int dj[3] = { 0, 1, 1 };

    while (i < rows && j < cols) {
        sum += arr[i][j];

        int next_val, best_i = -1, best_j = -1;

        for (int d = 0; d < 3; ++d) {
            int ni = i + di[d], nj = j + dj[d];

            if (ni < rows && nj < cols) {
                if (best_i == -1 || next_val < arr[ni][nj])
                    next_val = arr[ni][nj], best_i = ni, best_j = nj;
            }
        }

        if (best_i == -1)
            break;
        i = best_i, j = best_j;
    }
    cout << sum << "\n";
```

- In last code we tried 3 positions
  - (i+1, j), (i, j+1), (**i+1, j+1**)
  - The shift from (i, j) is
  - (1, 0), (0, 1), (1, 1)
- What if we coded the shifts in 2 arrays di, dj and used them
  - Then we stop all this copy/paste
- This is called **direction array**
  - Simple trick for cleaner code when u want to move to your **neighbours**

Practice: Flatten array

- Let Say we have matrix of ROWS x COLS
  - 1D here: 8 16 9 52 3 15 **<u>27</u>** 6 14 25 2 10
- To convert from (i, j) in matrix to 1D array
  - i * COLS + j
  - (1, 2) ⇒ 1 * 4 + 2 = 6
- To convert from index in 1D array to (i, j) in matrix
  - i = idx/COLS, j = idx%COLS
  - Idx = 6 ⇒ (6/4, 6%4) = (1, 2)
  - Why? Idx = i * COLS + j
    - Idx / COLS   = (i * COLS + j)/COLS   = i + 0, as j < COLS
    - Idx % COLS = (i * COLS + j)%COLS = 0 + j, as j < COLS and (i*COLS)%COLS = 0



8	16	9	52
3	15	27	6
14	25	2	10

“Acquire knowledge and impart it to the people.”

“Seek knowledge from the Cradle to the Grave.”