```
// Anthony Pucci
// 8 Queens Problem

# include <iostream>
# include <iomanip>
# include <fstream>

# define PATH "E:/4) Senior Year/2nd Semester/senior seminar/8 Queens Program/output-4.txt"
using namespace std;

void main()
{
	// Variable for file
	ofstream file;

	// Opening file
	file.open(PATH);

	cout << "Anthony Pucci" << endl;
	cout << "8 Queens Problem" << endl << endl;

	file << "Anthony Pucci" << endl;
	file << "8 Queens Problem" << endl << endl;

	// Variables
	int n = 0;				// Size of the board in [n x n] form

	int rowTest = 0;
	int rowComp = 0;
	int colTest = 0;
	int colComp = 0;
	int temp = 0;				// Used to set up board[n] equal to 0
	int locCounter = 0;			// Used to bring values from board[n] to printBoard[83]

	int solNum = 0;				// Solution number
	int current = 0;			// The column currently moving
	int test = 0;				// The column being compared with the moving column

	// Variables for horizontal print out
	int solCounter = 1;			// current solution it is up to in the print out
	int currentLocation = 0;		// current location in the solution array
	int i = 0;				// counter for the 6 solutions per line
	int j = 0;				// keeps track of where it is in solutions
	int loc = 0;
	int storeLoc = 0;			// current location to store board array in solutions
array
	int tracker = 0;			// tracks how many solutions have been put into the
solutions array

	int row = 0;
	int col = 0;
	int check = 0;

	//int solutionsAccross = 84/max(12, n) + 3;

	// Array for horizontal print out
	int solution[83] = {};			// solutions array

	cout << "Please enter in the size of the NxN board: ";
	cin >> n;
	cout << endl << "n: " << n << endl << endl;
	file << "n: " << n << endl << endl;

	int jump = 14 - n;			// jumps to the beginning of the next solution
```

```
int* board = new int[n];

if (n < 3)
{
	cout << "The board must be at least a 4x4!" << endl;
} // if (n < 3)

//sets array to 0
while (temp < n)
{
	*(board + temp) = 0;
	temp = temp + 1;
} // while (temp < n)

current = current + 1;

// --------- Collision Detection ---------

while (current != -1)
{
	rowTest = *(board + test);
	rowComp = *(board + current);

	colTest = test;
	colComp = current;

	// To prevent it from comparing to itself
	if (current == test)
	{
		current = current + 1;
		test = 0;

	} // if (current == test)

	// ---------- ROW ----------

	else if (*(board + current) == *(board + test))
	{
		test = 0;

		if ((board[current] + 1) > (n - 1))
		{
			while (board[current] == (n - 1))
			{
				current = current - 1;
				*(board + (current + 1)) = 0;
			} // while (board[current] == (n - 1))

			board[current] = board[current] + 1;
		}
		else
		{
			board[current] = board[current] + 1;
		} // else

		test = 0;

	} // else if (*(board + current) == *(board + test))

	// ---------- DIAGONAL ----------

	else if (abs(rowTest - rowComp) == abs(colTest - colComp))
	{
```

```
		test = 0;

		// goes past the last row
		if ((board[current] + 1) > (n - 1))
		{
			while (board[current] == (n - 1))
			{
				current = current - 1;
				*(board + (current + 1)) = 0;
			} // while (board[current] == (n - 1))

			board[current] = board[current] + 1;

		} // if ((board[current] + 1) > (n-1))

		// still on board
		else
		{
			board[current] = board[current] + 1;
		} // else

	} // else if(abs(rowTest - rowComp) == abs(colTest - colComp))

	// ------- NO COLLISION -------
	else
	{
		test = test + 1;

		if ((current == (n - 1)) && (test == (n - 1)))
		{
			solNum = solNum + 1;

			// ---------------------- Print out ----------------------

			// store the solution in the solution array
			while (loc < n)
			{
				solution[storeLoc] = *(board + loc);
				loc = loc + 1;
				storeLoc = storeLoc + 1;
			} // while (loc < n)

			loc = 0;
			tracker = tracker + 1;

			// Prints out the solution numbers and the actual solutions
			if (tracker == 6)
			{
				// Prints out the solution numbers
				i = 0;
				while (i < 6)
				{
					file << "Solution " << setw(5) << left << solCounter;
					solCounter = solCounter + 1;
					i = i + 1;
				} // while (i < 6)

				file << endl;

				// Board representation of the solutions
				row = 0;
				col = 0;

				while (row < n)
```

```
                        {
                                while (col < n*i)
                                {
                                        while (check < n)
                                        {
                                                if (solution[col] == row)
                                                {
                                                        file << "Q";
                                                } // if (solution[col] == row)
                                                else
                                                {
                                                        file << "-";
                                                } // else

                                                col = col + 1;
                                                check = check + 1;
                                        } // while (check < n)

                                        j = 0;

                                        // Prints out the spaces between the solutions
                                        while (j < jump)
                                        {
                                                file << " ";
                                                j = j + 1;
                                        } // while (j < jump)

                                        j = 0;
                                        check = 0;
                                } // while (col < n*i)

                                file << endl;
                                col = 0;
                                row = row + 1;
                        } // while (row < n)

                        // Prints out the solutions
                        while (currentLocation != n*i)
                        {
                                while (j < n)
                                {
                                        file << solution[currentLocation];
                                        currentLocation = currentLocation + 1;
                                        j = j + 1;
                                } // while (j < n)

                                j = 0;

                                // Prints out the spaces between the solutions
                                while (j < jump)
                                {
                                        file << " ";
                                        j = j + 1;
                                } // while (j < jump)

                                j = 0;
                        } // while (currentLocation != n*i)

                        file << endl << endl;

// Resets everything to 0 so it can restore next solutions in the array

                        // reset array to 0
                        while(temp < n*i)
```

```
                        {
                                solution[temp] = 0;
                                temp = temp + 1;
                        } // while(temp < n*i)
                        temp = 0;

                        //solution[83] = {};
                        tracker = 0;
                        i = 0;
                        loc = 0;
                        storeLoc = 0;
                        currentLocation = 0;

                } // if (tracker == 6)


                while (board[current] == (n - 1))
                {
                        current = current - 1;

                        *(board + (current + 1)) = 0;
                        test = 0;
                } // while (board[current] == (n - 1))

                board[current] = board[current] + 1;
                test = 0;
        } // if ((current == (n - 1)) && (test == (n - 1)))

    } // else

} // while (current != -1)

if (tracker < 6)
{
    // Prints out the solution numbers
    while (solCounter != (solNum + 1))
    {
        file << "Solution " << setw(5) << left << solCounter;
        solCounter = solCounter + 1;
    } // while (solCounter != (solNum + 1))

    file << endl;
} // if (tracker < 6)

// Board representation of the solutions
row = 0;
col = 0;

    while (row < n)
    {
        while (col < n*tracker)
        {
            while (check < n)
            {
                if (solution[col] == row)
                {
                    file << "Q";
                } // if (solution[col] == row)
                else
                {
                    file << "-";
                } // else

                col = col + 1;
```

```
                                check = check + 1;
                        } // while (check < n)

                        j = 0;

                        // Prints out the spaces between the solutions
                        while (j < jump)
                        {
                                file << " ";
                                j = j + 1;
                        } // while (j < jump)

                        j = 0;
                        check = 0;

                } // while (col < n*tracker)

                file << endl;
                col = 0;
                row = row + 1;

        } // while  (row < n)

        // Prints out the solutions
        while (currentLocation != n*tracker)
        {
                while (j < n)
                {
                        file << solution[currentLocation];
                        currentLocation = currentLocation + 1;
                        j = j + 1;
                }

                j = 0;

                // Prints out the spaces between the solutions
                while (j < jump)
                {
                        file << " ";
                        j = j + 1;
                }

                j = 0;

        } // while (currentLocation != (n*tracker))

        file.close();
        delete board;

} // main
```