| Hands-On Activity 5.1 | |
|---|---|
| **Multidimensional Arrays** | |
| **Course Code:** CPE007 | **Program:** Computer Engineering |
| **Course Title:** Programming Logic and Design | **Date Performed: 30/09/2025** |
| **Section: CPE11S1** | **Date Submitted: 30/09/2025** |
| **Name(s): James Daniel M. Verano** | **Instructor: Engr. Jimlord M. Quejado** |

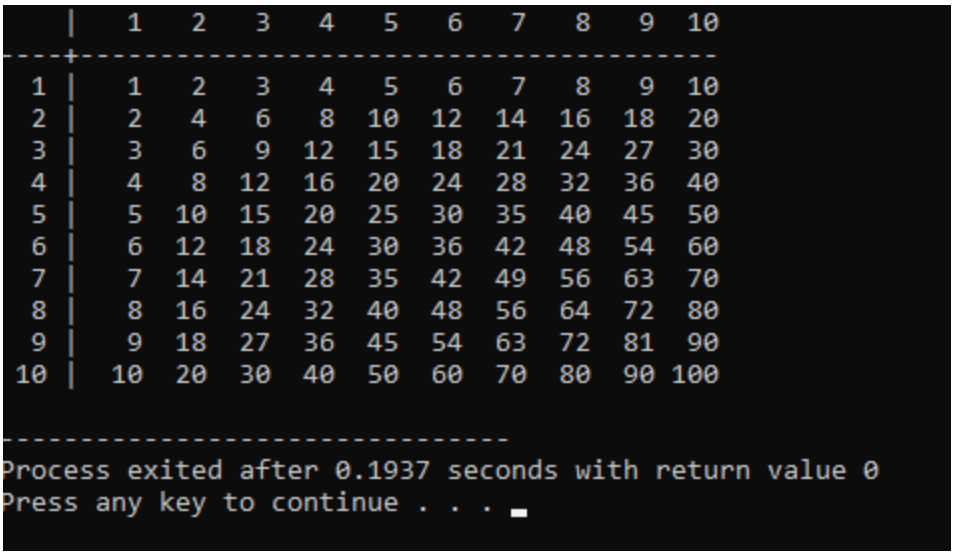
**6. Output**

**Example 1:**

**Code Snippet:**

```cpp
#include <iostream>
#include <iomanip>
using namespace std;

int main() {
    const int rows = 10;
    const int cols = 10;
    int table[rows][cols];

    // Fill the multiplication table
    for (int i = 0; i < rows; i++)
        for (int j = 0; j < cols; j++)
            table[i][j] = (i + 1) * (j + 1);

    cout << "    |";
    for (int j = 1; j <= cols; j++)
        cout << setw(4) << j;
    cout << "\n----+";
    for (int j = 0; j < cols; j++)
        cout << "----";
    cout << endl;

    for (int i = 0; i < rows; i++) {
        cout << setw(3) << (i + 1) << " |";
        for (int j = 0; j < cols; j++)
            cout << setw(4) << table[i][j];
        cout << endl;
    }

    return 0;
}
```

**Output:**




**Analysis:**

In the first for loop function, it is responsible for generating the multiplication values function and storing them in a two-dimensional array. The outer for loop function is the generating of rows where the 1 - 10 values will be used a multiplied in one another to get the other values, and same with the inner for loop function where it is the generating of columnar values of 1 - 10. Under the inner for loop function there is a processing of the product of [i + 1] * [j + 1], which is the process of getting the multiplication of each i and j value as it increments.

The second and third for loop functions are created only for the design structure of the multiplication table. The second for loop is mostly focused on the top row or column headers, that are showing the numbers 1 to 10. The loop (for (int j = 1; j <= cols; j++)) then prints each column number, from 1 to 10. In the third for loop, this is the design format for the horizontal dashed line, where it visually separates the headers from the body of the table. As you can see, we used the setw(4), the setw(4) belongs to the library extension of <Iomanip> which is for the format of aligning the numbers within a 4-character width.

In the final for loop function, this is where the first for loop comes back again, we remember that we had a code process of the product function of i and j, notated as, [i + 1] * [j + 1]. Now we will be printing them out. The outer for loop function, declared as for (int i = 0; i < rows; i++), this function handles each row of the table, which is the row number notated (i + 1). lastly, the inner for loop for (int j = 0; j < cols; j++) this function values will print out the previously stored values in table[i][j], which was the initialized function notated as, [i + 1] * [j + 1]. again, we used setw(4) for a cleaner alignment. This identifies that once the values of each column is printed in that one row, a newline will move to another row which loops up til we end up to a multiplication table.

**Example 2:**

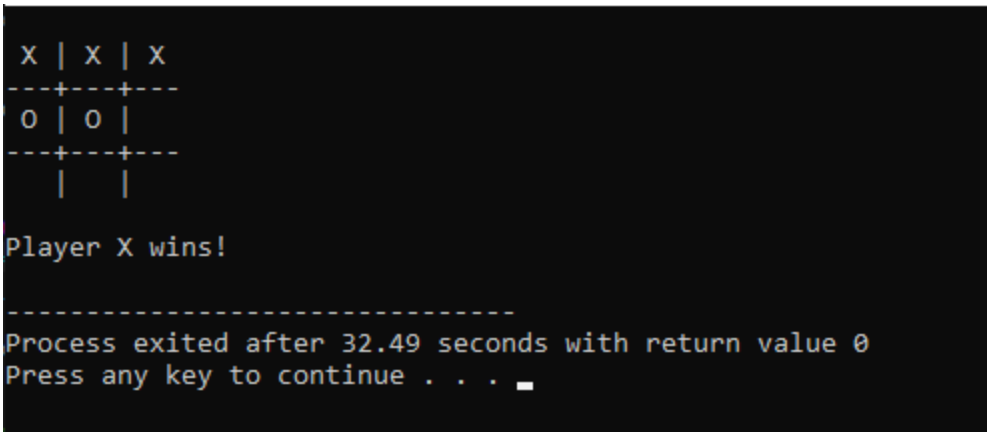
**Code Snippet:**

```cpp
#include <iostream>
using namespace std;

const int SIZE = 3;

void clear() {
    system("CLS");
}

void printBoard(char board[SIZE][SIZE]) {
    cout << "\n";
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            cout << " " << board[i][j];
            if (j < SIZE - 1) cout << " |";
        }
        cout << "\n";
        if (i < SIZE - 1) cout << "---+---+---\n";
    }
    cout << "\n";
}

// Check for a win possibilities
bool Winner(char board[SIZE][SIZE], char player) {
    // Checking for row / horizontal section, if the player takes the similar board[i] value
    // of the board, with each column number, this declarations turns true.
    for (int i = 0; i < SIZE; i++) {
        if (board[i][0] == player && board[i][1] == player && board[i][2] == player)
            return true;
    }
    // Checking for column / vertical section, if the player takes the similar board[i] value
    // of the board, with each row number, this declarations turns true.
    for (int j = 0; j < SIZE; j++) {
        if (board[0][j] == player && board[1][j] == player && board[2][j] == player)
            return true;
    }
    // Same with diagonals sections Approach in according to the corresponding given number.

    if (board[0][0] == player && board[1][1] == player && board[2][2] == player)
        return true;
    if (board[0][2] == player && board[1][1] == player && board[2][0] == player)
        return true;

    return false;
}

```

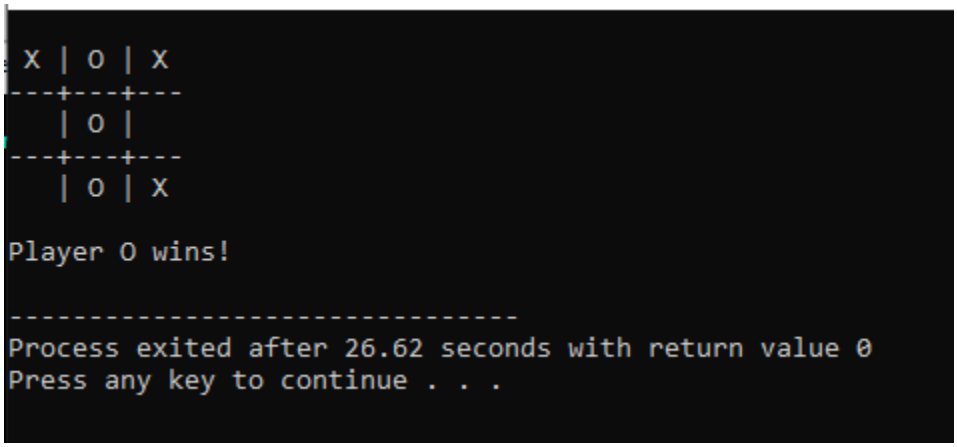
```
int main() {
    char board[SIZE][SIZE];

    // Initialize board with spaces
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            board[i][j] = ' ';
        }
    }

    int moves = 0;
    char player = 'X';
    int row, col;

    printBoard(board);

    while (moves < 9) {
        cout << "Player " << player << ", enter row and column (0-2): ";
        cin >> row >> col;

        if (row < 0 || row >= SIZE || col < 0 || col >= SIZE) {
            cout << "Out of range. Try again.\n";
            continue;
        }

        if (board[row][col] != ' ') {
            cout << "Cell occupied. Try again.\n";
            continue;
        }

        clear();

        board[row][col] = player;
        moves++;
        printBoard(board);

        if (Winner(board, player)) {
            cout << "Player " << player << " wins!\n";
            return 0;
        }

        player = (player == 'X') ? 'O' : 'X';
    }

    cout << "It's a draw!\n";
    return 0;
}
```

**Output:**

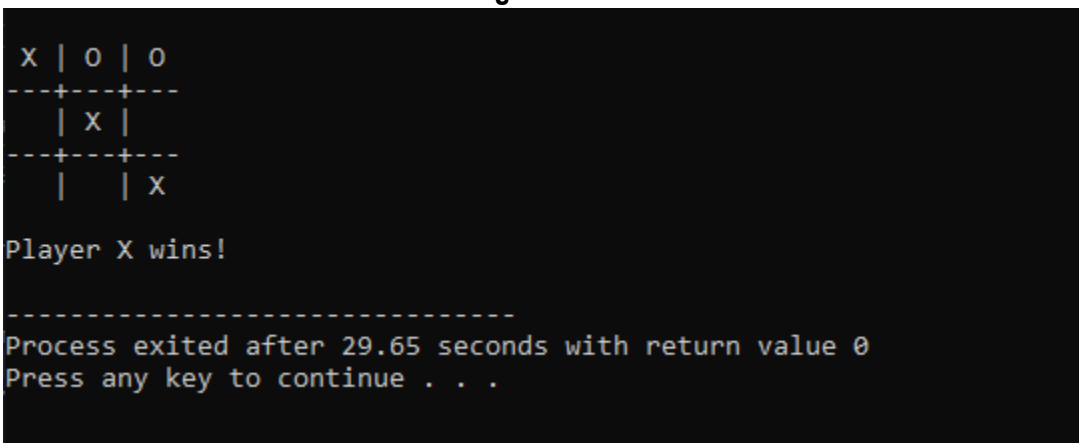
**Horizontal Win:**




**Vertical Win:**




**Diagonal Win:**



**Analysis:**

In this activity, we are tasked to create a tic tac toe game out of c++ program using Multidimensional Arrays. In the first functional line code, which is line code 4, is our constant variable declaration for our ***[SIZE]*** variable. The ***[SIZE]*** variable is what holds the positioning of the ***[i]*** and ***[j]*** variable value. The [i] and [j] value are for the inputs of where the player's designated symbols will be inputted. They are known as the row and column section too to be exact.

For Visuals:

***[i][j] = [ROW][COLUMN] = [SIZE]*** ⇒ Limits it from 0 - 2 as i and j is declared to 0.
[0][1] takes the first row, middle column spot.

On the next line code, which is the line 6 - 8, It is for a cleaner approach whenever we start another input. This is only for us to play tic tac toe clean and does not repeat on printing the table over and over again.

Now, In the next line code, which is the line 10 - 21, is the printing of the tic tac toe board format. It is just simple for loop functions that structures the position of the row and columns, and a visual design for cleaner game visuals.

For Visuals:

[ i = 0; i < SIZE; i++ and j = 0; j < SIZE; j++] ⇒ Implies to print out a space for the board[i][j]
and the[ j < SIZE - 1 ] ⇒ A structure that separates them from combining another, and also a cleaner visuals for row and column.

The next part of the code is the very important one, which is the line code 24 - 45. This line code function is the function where the variable ***[Winner]*** identifies different types of possibilities of winning positions. In the first for loop function, is the checking for row variable [i] / horizontal section, if the player takes the similar board[i] of the board, with each corresponding column number given in the if function, the for loop's function declaration will return true.

For Visuals:

board[i][0] ⇒ player == board[i][1] ⇒ player == board[i][2] ⇒ player (where [ i = 0 ])
board[0][0] ⇒ player == board[0][1] ⇒ player == board[0][2] ⇒ player = Returns true, win.
this means, row 0 with column 0 - 2, which means, x | x | x, that's a horizontal win.

In the second for loop function, it is just the same function of the first for loop but in column variable [j]. It is the checking for column / vertical section, if the player takes the similar board[j] of the board, with each corresponding row number given in the if statement function, this for loop's function declaration will return true.

For Visuals:

board[0][j] ⇒ player == board[1][j] ⇒ player == board[2][j] ⇒ player (where [ i = 0 ])
board[0][0] ⇒ player == board[0][1] ⇒ player == board[0][2] ⇒ player = Returns true, win.
this means, column 0 with row 0 - 2, which means:

| x |
| x |
| x |

This represents a vertical win.

In the last function codes, we used a if statement where IF the board is equal to [ i = j ] in all boards, that represents a diagonal win, and just a vise versa of the diagonal where in you takes both [ i and j ] index value of 0, and the value of [ i and j ] index 1.

for visuals:
board[i][j] ⇒ player == board[i][j] ⇒ player == board[i][j] ⇒ player = [i = j] where i and j is equal to 1 , 2 , 3
board[0]0j] ⇒ player == board[1][1] ⇒ player == board[2][2] ⇒ player = represents a diagonal win of:

| X | B | B
| B | X | B
| B | B | X
B = blank.

[ i and j ] index value of 0 and 2 vise versa and value of [ i and j ] index 1.
board[0][2] ⇒ player == board[1][1] ⇒ player == board[2][0] ⇒ player

If the function does not meet any of the for loop functions inside with if statements, it will return false, or draw.

Now, since we are now done in the outside block of the main code, we are going to proceed inside the main block code function. In the first for loop function, as what mentioned above in the line code 10 - 21, it simplifies where the player's designated symbol will be inputted.

We can see in line 57 - 59, we have declarations, where int moves = 0,
char player = 'X' [known as the designated symbol],
int row, col; where the upper function controls the row and column input value and if statements of possible winning position.

the line code 61, just prints the line code 10 - 21, which was the board format.

In the line code 63 to 89, is the final code. What does it do? This code is where now the loopings and also the switch of player's designated symbols and printing of symbols in their desired row and column input. ***WHILE (moves < 9)*** , knowing that after 9 moves, there's no longer any spots, it may potentially lead to either a draw or a final symbol input win. the rest of the inside function is more focused on IF the row and column are either LESS THAN 0, GREATER OR EQUAL THAN SIZE, it will print out a specific functioning line code.

And another if functioning code, where if [row][column] is not in the desired SIZE range, which means, it is not in between the value of 0 - 2, it will print out a specific line code, and it will still continue.

the clear() function is what clears the repetitive function of switching players and inputting the designated symbols. in the next line code, if a player inputs a valid [row][col], it increments the moves variable. In the final point, if the function has met some of the functional if statements that WERE inside the winner variable function, the one that was declared outside the main block, it will return a line code where the player wins. The line 88 is just the switching of the player's designated symbol. and that's all!

**7. Supplementary Activity**

- Screenshot of Code:
- Output of Code (label and compile ALL possible outputs):
- Short Analysis(min 5 sentences):

**8. Conclusion**

In my overall analysis and procedures, I was able to fully execute and identify each of the code's functionality and main purpose with regards to their correlation to one another. Given examples, such as the multiplication table, I learned how to store and generate values in a two-dimensional array. I was also introduced to the usage of the library extension *setw(4)*, which is used for cleaner format approach and better readability. And for the tic tac toe, it helped me to grasp a lot in multidimensional arrays and just simple arrays combined with for loop functions. The tic tac toe was one of the hardest programs that I have ever tried to understand and read in terms of coding analysis, yet it was bearable and I was able to get a grasp of it in the end, and descriptively explain it with all the effort I have. I believe I performed well in implementing the programs, but I also believe that I still have to improve myself in looking for a better and optimized code structure. This activity has made me improve my logical thinking in designing functional and interactive programs.