

C assignment - Adam Clarke

by Adam Clarke

Submission date: 05-Apr-2019 01:05PM (UTC+0100)

Submission ID: 104190687

File name: 462046666.C_Assignment.docx (465.25K)

Word count: 3430

Character count: 17861



SOFTWARE DEVELOPMENT

ACCA4017



JANUARY 1, 2019

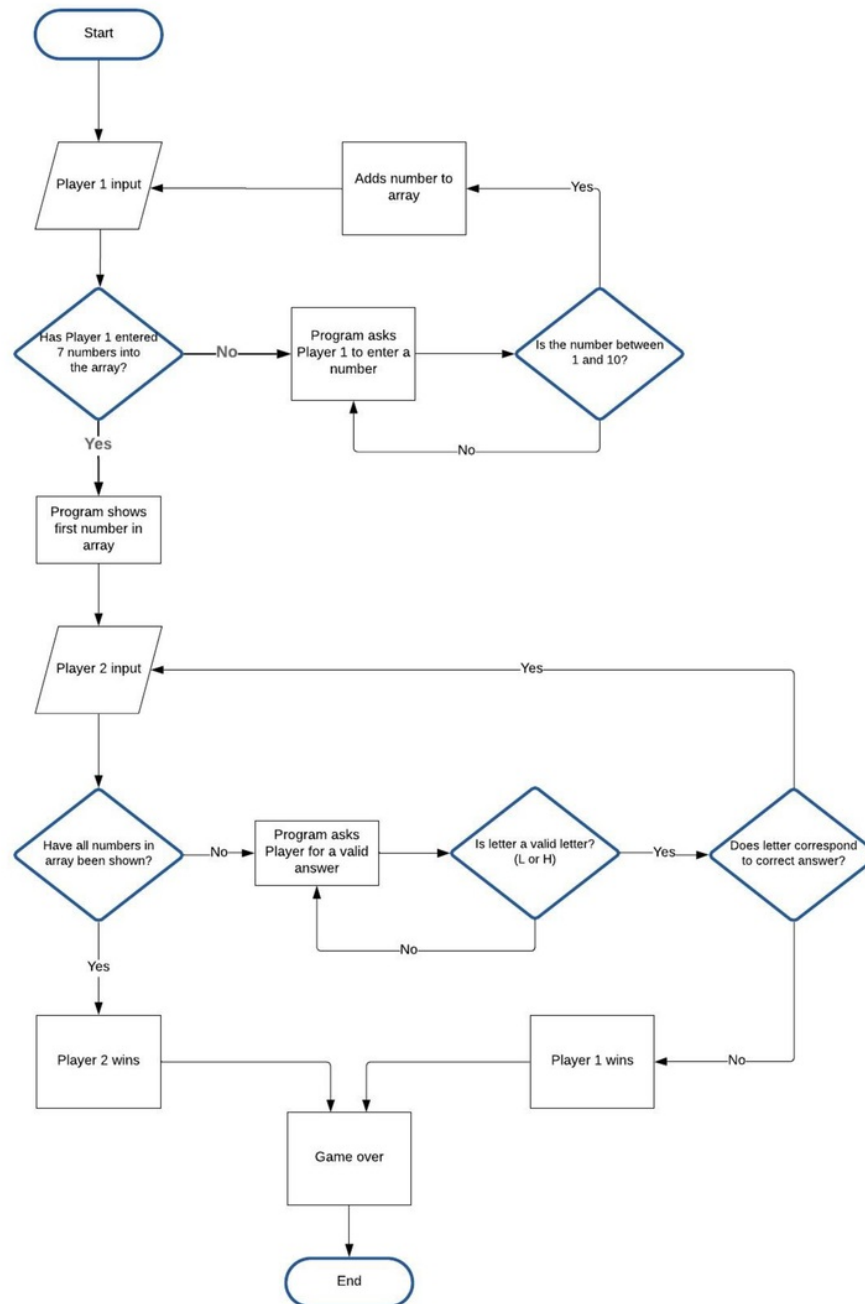
ADAM CLARKE

Contents

Stages of development:	2
Exercise 1:	2
Exercise 2:	4
Exercise 3:	7
Problems solved:	9
Exercise 1:	9
Exercise 2:	10
Exercise 3:	11
Tests conducted:	12
Exercise 1:	12
Exercise 2:	13
Exercise 3:	14
Appendix	14
Exercise 1:	14
Exercise 2:	16
Exercise 3:	18

Stages of development:

Exercise 1:



My first task was to create an array with 7 values where the values are inputted by the user (player 1). The screenshot below to the left depicts this stage of development; it also shows it is working correctly.

 C:\Windows\system32\cmd.exe

```
Player 1 enter 7 numbers
1
2
3
4
5
6
7
Press any key to continue . . .
```

To the right is a screenshot of the second stage of my project; storing the user input values into an array and then creating a loop that outputs all the user inputs one by one. At this point the player 2 input is obsolete. The loop runs regardless of what they enter.

```
The Number is 1
Higher (H) or lower(L)
h
Correct! Next Number
The Number is 2
Higher (H) or lower(L)
l
Incorrect! Player 1 Wins
```

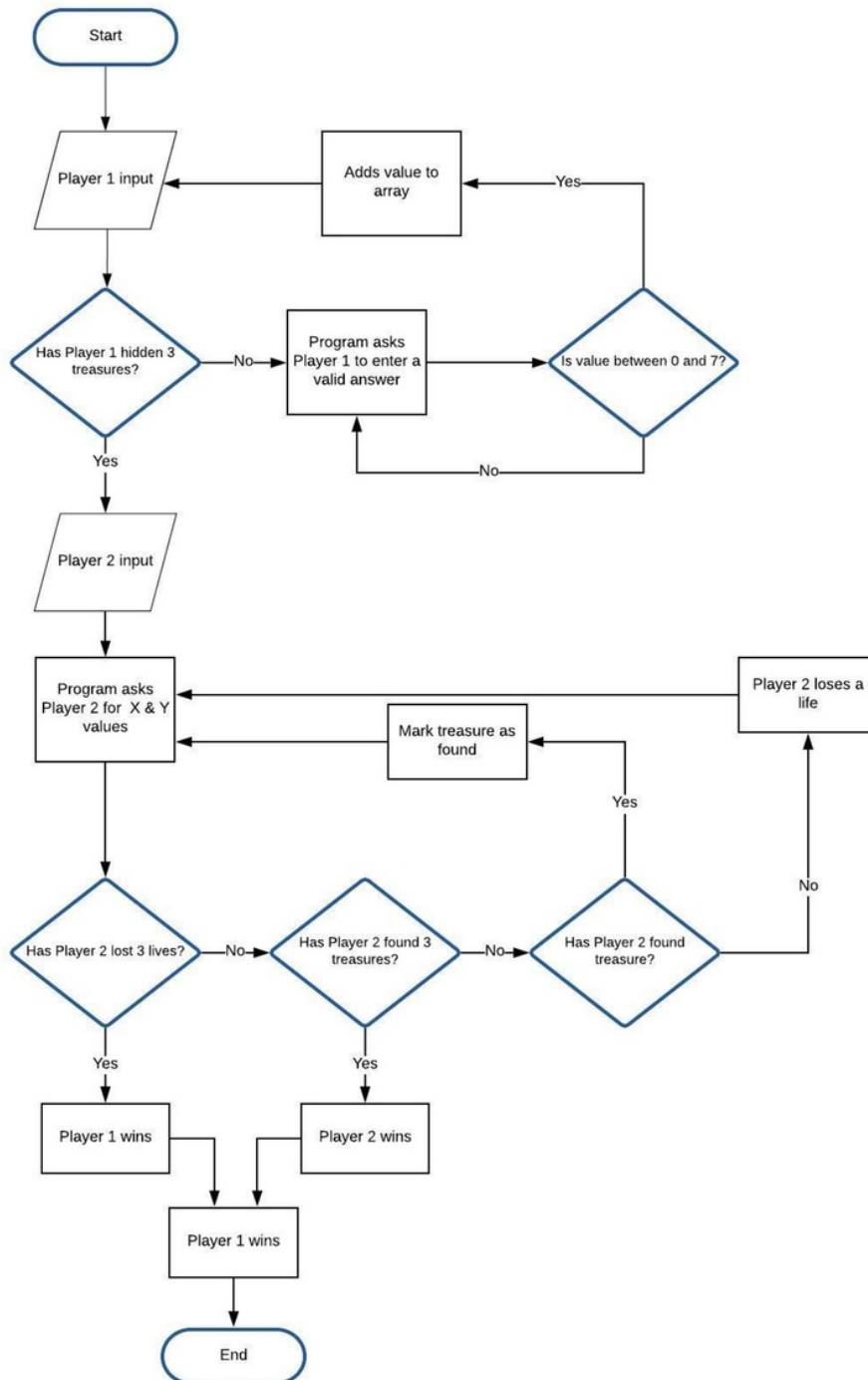
```
Player 1 enter 7 numbers
1
2
3
4
5
6
7
The Number is 1
Higher (H) or lower(L)
h
The Number is 2
Higher (H) or lower(L)
l
The Number is 3
Higher (H) or lower(L)
h
The Number is 4
Higher (H) or lower(L)
l
The Number is 5
Higher (H) or lower(L)
h
The Number is 6
Higher (H) or lower(L)
l
The Number is 7
Higher (H) or lower(L)
h
Press any key to continue . . .
```

Once I got the Player 1 input values to be stored in an array correctly, the next stage was ensuring that the Player 2 input was validated, and the game continued accordingly to the Player 2 input. For example, if Player 2 types "h" and the next value is higher then the game will progress but if the player types "h" and the next value is lower then the game will end and Player 1 will win. The screenshot above to the left shows this process.

```
Player 1 enter 7 numbers between 1 & 10
1
6 numbers left!
1
Error, cannot be same as previous!
6 numbers left!
100
Error, try a different number!
6 numbers left!
```

Above is one of the final stages of development; ensuring the Player 1 input is correct. To develop this criterion, it is required the Player 1 input values be an integer between 1 & 10, not be the same as the previous value and reject any input that does not match that criteria. After this, user testing was started.

Exercise 2:



The first task for exercise 2 was to create a 2D array and present it onscreen to look like a grid. This was stage 1:

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
Press any key to continue . . .
```

The next stage was to show Player 2 input being entered into the 2D array and then printed onscreen. This was stage 2 and is shown in the screenshot below:

```
Player 1 - Hide the treasure
Enter X & Y values for hidden treasure
X:1
Y:1
Player 1 - Hide the treasure
Enter X & Y values for hidden treasure
X:2
Y:2
Player 1 - Hide the treasure
Enter X & Y values for hidden treasure
X:3
Y:3

-----
-X-----
--X-----
---X-----
-----
-----
-----
-----
-----
```

It can be noted that at this point the array was changed from an integer variable to a character variable as this would allow 'X' & 'N' to be stored more efficiently.

```

Player 2 - Your Turn
Guess column across? 0-7: 1
Guess row down? 0-7: 1
Found some treasure at (1, 1)

Guess column across? 0-7: 2
Guess row down? 0-7: 2
Found some treasure at (2, 2)

Guess column across? 0-7: 3
Guess row down? 0-7: 3
Found some treasure at (3, 3)

All Treasure has been found!

Player 2 Wins!
-----
-X-----
--X-----
---X-----
-----
-----
-----
-----
-----
Press any key to continue . . .

```

The next stage was to implement input from Player 2 and match it against input from Player 1 to see if they have guessed correctly. Player 2 has 3 lives and is able to keep guessing the treasure locations until they run out of lives or they guess all 3 locations correctly. For every incorrect guess Player 2 will lose a life.

One of the final stages was to make the array more aesthetically pleasing when printed for Player 2 to see. It is clear from the screenshot to the right that it is much easier to read. A future improvement for this game could be to include the X & Y axes to further improve readability.

```

All Treasure has been found!

Player 2 Wins!
- - - - -
-  X  - - - -
-  -  X  - - -
-  -  -  X  - -
-  -  -  -  - -
-  -  -  -  - -
-  -  -  -  - -
-  -  -  -  - -
-  -  -  -  - -

```

Below shows the implementation of the final stage of development; ensuring that incorrect guesses are stored in the array as an N and then printed onscreen.

```

Player 1 Wins!
- - - - -
-  X  - - - -
-  -  X  - - -
-  -  -  X  - -
-  -  -  -  N  -
-  -  -  -  -  N
-  -  -  -  -  N  -
-  -  -  -  -  -

```

Exercise 3:

The first task for exercise 3 was to create and display the board. This was done using a similar method to Exercise 2; an array was created and then have each value stored as hyphen. When the players start to select columns to place their counters, the hyphens will then change accordingly.

```

- - - - -
- - - - -
- - - - -
- - - - -
- - - - -
- - - - -
1  2  3  4  5  6

```

The second step in creating the Connect 4 style game was to integrate the Player input; the Player entering an integer from 0-5 would correlate to which column the counter drops into. However, it can be seen that the counter does not drop to the bottom of the column as it should.

```

Place your first counter
Column: 4
- - - - X -
- - - - -
- - - - -
- - - - -
- - - - -
- - - - -
0  1  2  3  4  5

```

Now that the Player input has been integrated, the step after this would be to get the counter to drop to the bottom row of the column if there is not any counters below it.

```

Place your first counter
Column: 4
Please enter a valid number
- - - - -
- - - - -
- - - - -
- - - - -
- - - - X -
- - - - -
0  1  2  3  4  5

```

```

Place your first counter
Column: 1
- - - - -
- - - - -
- - - - -
- - - - -
- X - - -
0 1 2 3 4 5
Column: 1
- - - - -
- - - - -
- - - - -
- - - - -
- X - - -
- X - - -
0 1 2 3 4 5

```

The next stage of development was adding the ability to allow counters to stack on top of one another. All the user is required to do is choose a column and the game uses an "if" statement to determine whether or not the counter can be placed on the bottom row. As can be seen from the screenshot to the left, when two counters are dropped into the same column the second will be placed on top of the first.

When all spaces are filled within a column, the program will alert the Player they must choose another column.

```

- X - - -
- X - - -
- X - - -
- X - - -
- X - - -
0 1 2 3 4 5
Column: 1
Please enter a valid number
This Column is full - Choose another

```

This is only as far as the game has been developed. To further improve and develop this game, a second player could be added and their input displayed as an O on the board. Also, the game could detect whether or not 4 of the same counter have been placed in a row. A 'while' function could be used to check of this and have the rest of the game within this loop.

Problems solved:

Exercise 1:

```
n
The Number is 6
Higher (H) or lower(L)
l
The Number is 7
Higher (H) or lower(L)
h
Press any key to continue . . .
```

If Player 2 answers higher or lower correctly then they will get to keep guessing, however the program asks Player 2 to enter an answer (Higher or Lower) once the 7th value has been shown even though there is no value after 7 as shown in the screenshot to the right. To fix this – I allowed for the loop stop on the 6th value and then output the 7th outside of the loop. This fixed the issue.

The second problem encountered was Player 2 seeing the numbers that Player 1 previously entered. Considering this is a two player game – this would not be suitable so I have edited the code to include “std::system(“CLS”)” line of code. This line of code clears the screen of all previous input and output.

```
Player 1 enter 7 numbers between 1 & 10
1
6 numbers left!
2
5 numbers left!
100
4 numbers left!
```

One of the most difficult problems to solve was ensuring the Player 1 input was valid. As is clear from the screenshot to the left, the program would allow any number to be entered into the array. To resolve this problem, I used a switch statement to allow only

numbers between 1 & 10 to be allowed in the array – Values above and below this range would be rejected and not entered into the array. This also works for letters that are incorrectly entered.

Exercise 2:

```

Guess column across? 0-7: 6
Guess row down? 0-7: 6
Nothing there.

Guess column across? 0-7: 5
Guess row down? 0-7: 5
Nothing there.

Guess column across? 0-7: 3
Guess row down? 0-7: 3
Found some treasure at (3, 3)

```

All Treasure has been found!

Player 2 Wins!

```

-----
-X-----
--X-----
---X-----
-----
-----
-----
-----
-----

```

The first problem encountered was the map not showing values for incorrect guesses. For example, if Player 2 guessed coordinates 3, 4 and there is no treasure stored there, 'N' will appear on the map at 3, 4. If they guess correctly then 'X' will appear at 3, 4.

The problem was easily solved by inputting the Player 2 guess values as 'N' in the array. This can be shown below:

All Treasure has been found!

Player 2 Wins!

```

-----
-X-----
--X-----
---X-----
---N-----
---N-----
-----
-----
-----

```

The second problem encountered was Player 2 seeing the numbers that Player 1 previously entered. Considering this is a two player game – this would not be suitable so I have edited the code to include "std::system("CLS")" line of code. This line of code clears the screen of all previous input and output.

```

Enter X & Y values for hidden treasure (0-8)
X:1
Y:1
Player 1 - Hide the treasure
Enter X & Y values for hidden treasure (0-8)
X:1
Y:1
Player 1 - Hide the treasure

```

Another issue encountered was Player 1 being able to enter the same values as the previous treasure as can be seen in the screenshot above. Unfortunately, this issue is still apparent, however one way to fix this could be to insert a rule that doesn't allow user to enter previous values. This could be achieved by storing valid input of X and Y values into an array and then check Player 1 input using an "while" statement as follows:

```

while (playerInput = hiddenTreasure[0][0] || playerInput = hiddenTreasure[1][1]) {
    cout << "Can't be the same as previous" << endl;
    cin.clear();
    cin.ignore();
    cin >> playerInput;
}

```

The above code could be used as a future fix to this issue where "hiddenTreasure" is the array storing X & Y values and "playerInput" is the Player 1 input for the X and Y values.

Exercise 3:

```
Place your first counter
Column: 4
- - - - X -
- - - - - -
- - - - - -
- - - - - -
- - - - - -
- - - - - -
0 1 2 3 4 5
```

```
- - - - -
- - - - -
- - - - -
- - - - -
- - - - X -
0 1 2 3 4 5
Column: 4
- - - - -
- - - - -
- - - - -
- - - - -
- - - - X -
- - - - X -
0 1 2 3 4 5
```

The first problem encountered with Exercise 3 was the counters not dropping to the lowest point once the user chooses a column. The issue was resolved by inserting the counter to the bottom by default and then adding an “if” function that allows the counters to stack on the row above the previous. The “if” function will add the counter to the lowest point in the user’s desired column if there isn’t a counter already there.

The screenshot to the right shows that this feature is working, and the issue is now resolved.

```
X X X X X X
X X X X X X
X X X X X X
X X X X X X
X X X X X X
X X X X X X
0 1 2 3 4 5
Column: 5
Please enter a valid number
This Column is full - Choose another
```

Another problem that was encountered was all counters being placed on the board and the game not ending even though the Player cannot make any moves.

Tests conducted:

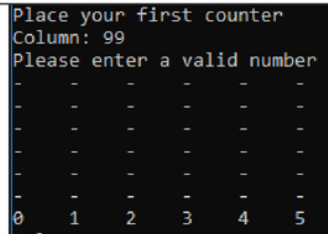
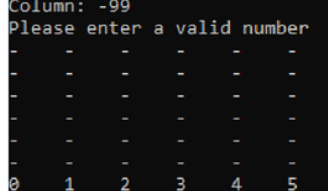
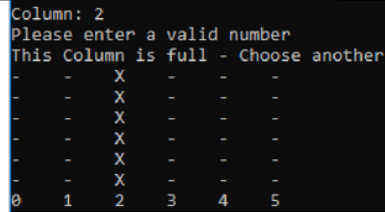
Exercise 1:

Test	How?	Result	Evidence
Player 1 input – Subsequent duplicate values.	Entering '1', then entering '1' again	The result prompts Player 1 for a new value. Loop is not continued.	Player 1 enter 7 numbers between 1 & 10 1 6 numbers left! 1 Error, cannot be same as previous! 6 numbers left!
Player 1 input – Entering a letter	Entering 'h'	The result prompts Player 1 for a new value. Loop is not continued.	Player 1 enter 7 numbers between 1 & 10 h Error, try a different value!
Player 1 input – Entering multiple letters	Entering 'hhhh'	The result prompts Player 1 for a new value over 4 lines. Loop is not continued.	Player 1 enter 7 numbers between 1 & 10 hhhh Error, try a different value! Error, try a different value! Error, try a different value! Error, try a different value!
Player 1 input – Entering values over the parameters	Entering '100'	The result prompts Player 1 for a new value. Loop is not continued.	Player 1 enter 7 numbers between 1 & 10 100 Error, try a different value!
Player 1 input – Entering values under the parameters	Entering '-100'	The result prompts Player 1 for a new value. Loop is not continued.	Player 1 enter 7 numbers between 1 & 10 -100 Error, try a different value!
Player 1 input – Loop ends after 7 values entered	Entering 7 valid values	Loop is ended and begins next stage of the game	1 number left! 7 The Number is 1 Higher (h) or lower(l)
Player 2 input – Player 2 wins	Correctly guessing all 7 values	Player 2 wins. Game ends.	The Final Number is 7 Player 2 Wins Press any key to continue . . .
Player 2 input – Player 1 wins	Incorrectly guessing a value	Player 1 wins. Game ends.	Higher (h) or lower(l) l Incorrect! Player 1 Wins Press any key to continue . . .
Player 2 input – Entering a letter (not L or H)	Entering 'k'	The result prompts Player 2 for a new value. Loop is not continued.	Higher (h) or lower(l) k Please enter a valid answer (h) or (l) The Number is 1 Higher (h) or lower(l)
Player 2 input – Entering a number	Entering '1'	The result prompts Player 2 for a new value. Loop is not continued.	Higher (h) or lower(l) 1 Please enter a valid answer (h) or (l) The Number is 1 Higher (h) or lower(l)

Exercise 2:

Test	How?	Result	Evidence
Player 1 input – Entering values over X parameters	Entering '99'	The result prompts Player 1 for a new value. Loop is not continued.	Player 1 - Hide the treasure Enter X & Y values for hidden treasure (0-8) X:99 Please enter a valid number (0-8) X:
Player 1 input – Entering values under X parameters	Entering '-1'	The result prompts Player 1 for a new value. Loop is not continued.	Player 1 - Hide the treasure Enter X & Y values for hidden treasure (0-8) X:-1 Please enter a valid number (0-8) X:
Player 1 input – Entering values over Y parameters	Entering '99'	The result prompts Player 1 for a new value. Loop is not continued.	Y:99 Please enter a valid number (0-8) Y:
Player 1 input – Entering values under Y parameters	Entering '-1'	The result prompts Player 1 for a new value. Loop is not continued.	Y: -1 Please enter a valid number (0-8) Y:
Player 1 input – Entering valid value for X	Entering '5'	The value is accepted and added to the array of coordinates.	Player 1 - Hide the treasure Enter X & Y values for hidden treasure (0-8) X:5 Y:
Player 1 input – Entering valid value for Y	Entering '5'	The value is accepted and added to the array of coordinates.	Enter X & Y values for hidden treasure (0-8) X:5 Y:5 Player 1 - Hide the treasure
Player 2 input – Entering correct value	Entering '1' as X and '1' as Y (once Player 1 has put treasure here)	Informs Player 2 they have "Found some treasure". Asks for next guess.	Player 2 - Your Turn Guess column across? 0-7: 1 Guess row down? 0-7: 1 Found some treasure at (1, 1)
Player 2 input – Entering incorrect value	Entering '4' as X and '4' as Y (No treasure put here by Player 1)	Informs Player 2 there is "Nothing there" and asks for another guess. Player 2 loses a life (although this doesn't display).	Guess column across? 0-7: 4 Guess row down? 0-7: 4 Nothing there. Guess column across? 0-7:
Player 2 Wins the game- Guesses 3 locations correctly	Guess all 3 locations correctly.	Informs Players that "Player 2 wins". Prints array as treasure map. Shows treasure locations as X and incorrect guesses as N. Game ends.	Player 2 Wins! - - - - - - X - - - - - X - - - - - X - - - - - N - - - - - - - - - - - - - - - -
Player 1 wins the game – Player 2 guesses 3 locations incorrectly	Guess all 3 locations incorrectly	Informs Players that Player 2 is "Out of guesses" & "Player 1 wins". Prints array as treasure map. Shows treasure locations as X and incorrect guesses as N. Game ends.	Out of guesses! Player 1 Wins! X - - - - - - X - - - - - X - - - - - - - - - - - - - - - - N - - - - - - N - - - - - - N

Exercise 3:

Test	How?	Result	Evidence
Player 1 input – Entering values over the parameters	Entering '99' as the column	Counter was not placed in any column and user input was cleared. User is told to "Please enter a valid number" and prompted to enter another value.	 <pre> Place your first counter Column: 99 Please enter a valid number - 0 1 2 3 4 5 </pre>
Player 1 input – Entering values under over the parameters	Entering '-99' as the column	Counter was not placed in any column and user input was cleared. User is told to "Please enter a valid number" and prompted to enter another value.	 <pre> Column: -99 Please enter a valid number - 0 1 2 3 4 5 </pre>
Player 1 input – Placing counter in column that is already full	Entering '2' after column 2 is already filled with counters	Counter was not placed in any column and user input was cleared. User is told to "Please enter a valid number" & "This column is full – Choose another" and prompted to enter another value.	 <pre> Column: 2 Please enter a valid number This Column is full - Choose another - - X - - - - X - - - - X - - - - X - - - - X - - - - X - - 0 1 2 3 4 5 </pre>

Appendix

Exercise 1:

```

#include "pch.h"
#include <iostream>

using namespace std;

int main()
{
    int playerNumbers[7], index = 0;
    char playerGuess;
    int guess;                                // This section declares all variables used

    cout << "Player 1 enter 7 numbers between 1 & 10" << endl;
    while (index < 7)
    {
        if (index != 0) {
            cout << 7 - index;
            cout << " number";
            if (index != 6) {
                cout << "s";
            }
            cout << " left!" << endl;        // Informs user of how many
            values they need to enter
        }
    }

```

```

cin >> guess;
switch (guess) {
case 1: // User can enter 1
case 2: // User can enter 2
case 3: // User can enter 3
case 4: // User can enter 4
case 5: // User can enter 5
case 6: // User can enter 6
case 7: // User can enter 7
case 8: // User can enter 8
case 9: // User can enter 9
case 10: // User can enter 10
    if (index > 0 && playerNumbers[index - 1] == guess) {
        cout << "Error, cannot be same as previous!" << endl; //
        Cannot enter value same as previous
        continue;
    }

    playerNumbers[index] = guess;
    index++; // Continues Incrementing
    break;

default:
    cout << "Error, try a different value!" << endl; //All
    other values will be incorrect
    break;
}

if (cin.fail()) {
    cin.clear();
    cin.ignore();
    continue; // If user doesn't enter a valid number,
    input is not stored
}

std::system("CLS"); // Clears all text on screen

for (index = 0; index < 6; index++) //Starts incrementing loop
{
    cout << "The Number is " << playerNumbers[index] << endl; //
    Prints numbers in the array (Player 1 input)
    cout << "Higher (h) or lower(l)" << endl;

    cin >> playerGuess;

    if (playerGuess == 'h' && playerNumbers[index] < playerNumbers[index -
1]) // Player 2 enters h & value is lower - Game over, Player 1 wins
    {
        cout << "Incorrect! Player 1 Wins" << endl;
        return(0);
    }

    else if (playerGuess == 'l' && playerNumbers[index] >
playerNumbers[index - 1]) // Player 2 enters l & value is higher - Game over,
Player 1 wins
    {
        cout << "Incorrect! Player 1 Wins" << endl;
    }
}

```

```

        return(0);
    }

    else if (playerGuess != 'h' && playerGuess != 'l')           // Answer
not accepted if it is not l or h
    {
        3 cout << "Please enter a valid answer (h) or (l)" << endl;
        index--;
    }

    else {
        3 cout << "Correct! Next Number" << endl;           // Player 2
guesses correct answer
    }

    }

    cout << "The Final Number is " << playerNumbers[6] << endl;           // Prints
final number in array
    cout << "Player 2 Wins" << endl;           // Game over - Player 2 wins!
    3 return 0;
}

```

Exercise 2:

```

#include "pch.h"
#include <iostream>

using namespace std;

const int NUMROWS = 8;
const int NUMCOLS = 8;
const int NUMLIVES = 3;
const int TREASURE = 3;           //Declares all constant
variables

int main()
{
    char map[NUMROWS][NUMCOLS];           // Treasure stored in array

    4 int row, col;

    for (row = 0; row < NUMROWS; row++)           // Map is cleared
        for (col = 0; col < NUMCOLS; col++)
            map[row][col] = '-';

    int inputRow, inputCol;

    for (int timesHidden = 0; timesHidden < TREASURE; timesHidden++)           // Starts
loop for treasure hidden 3 times
    {
        cout << "Player 1 - Hide the treasure" << endl;
        // Picks place on map to hide treasure
        cout << "Enter X & Y values for hidden treasure (0-8)" << endl;
        cout << "X:";
        cin >> inputCol;
        while (inputCol > 8 || inputCol < 0) {
            //If input is greater than 8 or smaller than 0 - Ask for another value
            cin.clear();
            cin.ignore();
        }
    }
}

```

```

        cout << "Please enter a valid number (0-8)" << endl;
        cout << "X: ";
        cin >> inputCol;
    }

    cout << "Y: ";
    cin >> inputRow;

    while (inputRow > 8 || inputRow < 0) {
        //If input is greater than 8 or smaller than 0 - Ask for another value
        cin.clear();
        cin.ignore();
        cout << "Please enter a valid number (0-8)" << endl;
        cout << "Y: ";
        cin >> inputRow;
    }

    inputRow == NUMROWS;
    inputCol == NUMCOLS;
    map[inputRow][inputCol] = 'X';
        // Marks coordinates as 'X' - Easily identifiable
}

std::system("CLS"); // Clears all Player 1 input

cout << "Player 2 - Your Turn" << endl;

int rowGuessed, colGuessed, treasureFound;
int guessesLeft = NUMLIVES;

treasureFound = 0;
while (guessesLeft > 0 && treasureFound < 3) //
Starts life counter (Player has 3 lives)
{
    cout << "Guess column across? 0-" << NUMCOLS - 1 << ": ";
    cin >> colGuessed;
        // Player 2 guesses where treasure is hidden
    cout << "Guess row down? 0-" << NUMROWS - 1 << ": ";
    cin >> rowGuessed;

    if (map[rowGuessed][colGuessed] == 'X') // Checks if
treasure is buried where Player 2 has guessed
    {
        cout << "Found some treasure at ("; // Mark
treasure as found

        cout << colGuessed << ", " << rowGuessed << ")" << endl;

        map[rowGuessed][colGuessed] = 'X';

        treasureFound++; //Treasure Found counter
increases
    }
    else
    {
        cout << "Nothing there." << endl; // If no treasure
is found - lose a life

        map[rowGuessed][colGuessed] = 'N';
        guessesLeft--;
    }
}

```

```

        cout << endl;

    }

    if (guessesLeft == 0) { // If Player 2 loses all lives - Player 1 wins
        cout << endl << "Out of guesses!" << endl;
        cout << endl << "Player 1 Wins!" << endl;
    }

    else
    {
        cout << endl << "All Treasure has been found!" << endl; // If all
        treasure is found - Player 2 wins
        cout << endl << "Player 2 Wins!" << endl;
    }

    for (row = 0; row < NUMROWS; row++) // Show map on
screen
    {
        1 for (col = 0; col < NUMCOLS; col++)
            cout << map[row][col] << " ";
        2 cout << endl;
    }

    return 0;
}

```

Exercise 3:

```

#include "pch.h"
#include <iostream>

using namespace std;

const int NUMROWS = 6;
const int NUMCOLS = 6; //Constant Variables

int main()
{
    char board[NUMROWS][NUMCOLS]; //Creates array with 6 rows & 6
columns
    1 for (int row = 0; row < NUMROWS; row++)
        for (int col = 0; col < NUMCOLS; col++)
            board[row][col] = '-'; //Sets all values
in array to '-'

    1 for (int row = 0; row < NUMROWS; row++)
    {
        for (int col = 0; col < NUMCOLS; col++)
            cout << board[row][col] << " "; //Creates spaces between
array values (improves readability)
        cout << endl;
    }

    cout << "0  1  2  3  4  5 " << endl; //Prints X axes
(Columnns)

    cout << "Place your first counter" << endl;
}

```

```

    int inputCol;

    for (int index = 0; index < 36; index++) { //Ends game after
        36 turns (36 is number of spaces on the board)

        cout << "Column: ";
        cin >> inputCol;

        if (inputCol != 1 || inputCol != 2 || inputCol != 3 || inputCol != 4 ||
            inputCol != 5 || inputCol != 0 ) { //Invalid numbers

            cout << "Please enter a valid number" << endl;
            index--;
        }

        while (inputCol >= 0 && inputCol <= 6) { //Places counter in
            bottom row
                if (board[5][inputCol] == '-') {
                    board[5][inputCol] = 'X';
                }
                else if (board[4][inputCol] == '-') { //Places counter
                    in next row up
                        board[4][inputCol] = 'X';
                }
                else if (board[3][inputCol] == '-') { //Places counter
                    in next row up
                        board[3][inputCol] = 'X';
                }
                else if (board[2][inputCol] == '-') { //Places counter
                    in next row up
                        board[2][inputCol] = 'X';
                }
                else if (board[1][inputCol] == '-') { //Places counter
                    in next row up
                        board[1][inputCol] = 'X';
                }
                else if (board[0][inputCol] == '-') { //Places counter
                    in next row up
                        board[0][inputCol] = 'X';
                }
                else {
                    cout << "This Column is full - Choose another" << endl;
                    //Alerts Player column is full
                    index--;
                    // Minus 1 from loop counter
                }
                break;
            }
        }

        1
        for (int row = 0; row < NUMROWS; row++)
        {
            for ( int col = 0; col < NUMCOLS; col++)
                cout << board[row][col] << " ";
            //Prints board with all values
            cout << endl;
        }
        cout << "0   1   2   3   4   5 " << endl;
    }
}

```

C assignment - Adam Clarke

ORIGINALITY REPORT

7%

SIMILARITY INDEX

5%

INTERNET SOURCES

1%

PUBLICATIONS

6%

STUDENT PAPERS

PRIMARY SOURCES

1

www.chegg.com

Internet Source

2%

2

Submitted to University of Lincoln

Student Paper

2%

3

Submitted to Pembrokeshire College

Student Paper

1%

4

Submitted to Contra Costa Community College District

Student Paper

1%

5

Submitted to Sim University

Student Paper

<1%

6

www.avantel.net

Internet Source

<1%

7

cvlab.epfl.ch

Internet Source

<1%

8

www.coursehero.com

Internet Source

<1%

Exclude quotes Off

Exclude matches Off

Exclude bibliography Off

C assignment - Adam Clarke

GRADEMARK REPORT

FINAL GRADE

67 / 100

GENERAL COMMENTS

Instructor

Exercise 1 38/ 50%

Flow chart shows logical planning of task and conforms to conventions taught, just ensure all paths have yes or no on them when coming from decision box. The original brief did not specify that player one could not repeat a number, only that they numbers they entered were between 1 and 10, however you have successfully implemented this feature. An if statement with the range `guess > 0 && guess < 8` would have been a more efficient check than a switch, and the use of functions could have aided development, maintenance and readability of code. Complete testing of ranges by using boundaries, eg test at 0, 1, 7 and 8, if you get successful results from these tests, you can be sure your code will be successful for all numerical input, although a test for a negative number and very large number is also useful.

Exercise 2 23/ 30%

Flow chart shows logical planning of task and conforms to conventions taught, it is almost logically correct (very end- both paths going to player 1 wins?) although you may also have considered testing player 2's input was also within the valid range (as you have already implemented this for player 1 you can reuse code- particularly if this is implemented in a function!)

Again as for previous exercise consider the use of functions.

Exercise 3 6/ 20%

You have not completed this exercise but have been able to implement some basic features, such as dropping counters until the column is full and checking column input is a valid one. Further marks would have been gained for completing this exercise.

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20
