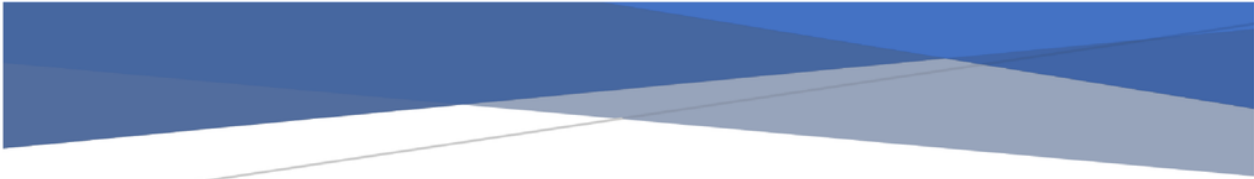


A1

by Luke Webb

FILE	12132_LUKE_WEBB_A1_178138_1987501756.DOCX (322.03K)		
TIME SUBMITTED	05-APR-2019 10:49PM (UTC+0100)	WORD COUNT	1372
SUBMISSION ID	104248499	CHARACTER COUNT	5458



ASSESSMENT
Software Development

Luke Webb
1803165

Exercise 1

Problem	Solution
<u>Having a way to handle if the game is on going or not</u>	<u>Having a bool to keep the game going so at the end of the game it can be changed to false to stop everything and end the game</u>

The first game is a very simple, its just a guessing game for if the next number will be higher or lower then the last number, this is what the stop of my code looks like, I define Print so that its faster for me to print out all the text and numbers I'm going to use, I also define variables that I'm going to use I'll get to the uses of them when they are used in my code.

```
#include <iostream>
#include <sstream>
#include <string>

//define Print as std::cout
#define Print(x) std::cout << x;

int main() {
    /* Define needed variables */
    bool game = true;
    bool once = false;
    std::string PlayerOneValues[7];
    std::string truth;
    std::string answer;
    std::string exit;
```

```
/* Start infinite loop */
while (game)
{
    /* Ask player 1 to input 7 numbers */
    Print("Player 1 please enter 7 numbers" << std::endl);

    /* loop 7 times for player 1 to input numbers */
    for (int i = 0; i < 7; i++){
        Print("Enter Number " << i + 1 << " > ");
        /* Assign player 1 input to array */
        std::getline(std::cin, PlayerOneValues[i]);
    }
}
```

The first thing that happens in the code is that there is a infinite loop so the game can keep going for as long as it need to or until the "game" is set to false in the code, the first line that prints out asks the first player for 7 numbers and then we have a for loop to get the 7 numbers from the player.

```

/* Loop another 7 times */
for (int i = 0; i < 7; i++) {
    /* if the variable once is false print the first number to the player*/
    if (once == false){
        Print("Player 2 the first number is " << PlayerOneValues[i] << std::endl);
        /* set once to true */
        once = true;
    }
    else {
        /* Ask player 2 if the number is higher or lower */
        Print("Higher (H) or Lower (L) > ");
        /* Assign player 2's input as answer */
        std::getline(std::cin, answer);

        /* check if the number is actually higher or lower*/
        if (PlayerOneValues[i - 1] < PlayerOneValues[i]) {
            truth = "H";
        }
        else if (PlayerOneValues[i - 1] > PlayerOneValues[i]) {
            truth = "L";
        }

        /* check player 2's input to the actual answer*/
        if (answer == truth) {
            Print("Correct, next value is " << PlayerOneValues[i] << std::endl);
        }
        else {
            Print("Incorrect, Player 1 Wins!" << std::endl);
            /* break from the loop and infinite loop */
            game = false;
            break;
        }
    }
}

```

Here is the main chunk of code that the game consists of, there is a for loop that loops 7 times asking the second player after display the first number if the next number is going to be higher or lower then the next, then in the middle there is code for checking the second players input on if they guess higher or lower or not and at the bottom we have the checks for if player 2 was correct or if player one won the game.

```

/* Ask player's if they want to play again */
Print("Want to Try Again? (Y) (N) > " << std::endl);
/* get players input */
std::getline(std::cin, exit);
/* check if the players input was yes*/
if (exit == "Y") {
    continue;
}
else {
    game = false;
}
}
/* pause cmd */
system("pause");
return 0;
}

```

The last lines in of the program look like this, at the top it asks if the player wants to play again taking in input for yes or no and handles what to do accordingly and the last lines pause the game so the user can read what's happening before the game closes for good.

Exercise 2

Problem	Solution
<u>Display the Grid with a index number on the top and bottom</u>	<u>Using a double for loops to iterate both detentions of an array at once</u>
<u>Stopping the user from enter numbers larger then 7 for the grids</u>	<u>If the number is larger than 7 player 1 will have to start again and player 2 will lose a life for entering a number not possible to be reached in game.</u>

In exercise 2 I thought the best way to display the grid was to have one for being displayed and one to have the information for the game in the background so I would have a 2D-Array of bools and chars so that I can display the chars and use the bools for testing what point the game is at

```
char    grid[8][8];
bool    tGrid[8][8];

int lives = 3;
int corretGuess = 0;
```

The next step was to display the grid of chars so that the players could see what was happening after every turn the best way to do this was to use a double for loop one loop for the x position and another for the y position.

```
void displayGrid() {
    std::cout << " ";

    for (size_t i = 0; i < 8; i++) {
        std::cout << i << " ";
        if (i >= 7) {
            std::cout << std::endl;
        }
    }

    for (size_t x = 0; x < 8; x++) {
        std::cout << x << " ";
        for (size_t y = 0; y < 8; y++) {
            std::cout << grid[x][y] << " ";
        }
        std::cout << std::endl;
    }
}
```

I also wanted to display the number of the column and row of the table, to get the top row I just did a for loop to count up to 8 displaying the number going right then I just needed to display the row number and the grid information using the double for loop, because I'll be using this code every time

I display the grid I put it into a function to stop repeating myself and after the same code written down a lot.

The next step to the game was to get player ones input and setup the default values for the grid that's going to be display after every turn, to get the first players input I made an array of 6 ints that is going to be used to hold 3 pairs of x and y coordinates.

```
unsigned int treasureLocations[6] = {0,0,0,0,0,0};

std::cout << "Player 1 place the treasure! " << std::endl;

std::cout << "x: "; std::cin >> treasureLocations[0];
std::cout << "y: "; std::cin >> treasureLocations[1];
std::cout << "Treasure 1 placed" << std::endl;
```

The initialized value of this array is set to 0 so that later on when I check if they entered valued information it doesn't crash the program, after getting the first set of x and y coordinates I need to check that the first player doesn't just enter the same information 2 more times because this will break the game, to check that the user doesn't do this I just checked the first 2 values of the array to the next 2 values and if they were the same this function would call it self resetting the input like this.

```
std::cout << "x: "; std::cin >> treasureLocations[2];
std::cout << "y: "; std::cin >> treasureLocations[3];
if (treasureLocations[0] && treasureLocations[1] == treasureLocations[2] && treasureLocations[3]) {
    std::cout << "You can't enter the same place twice! try again" << std::endl;
    setupGrid();
} else {
    std::cout << "Treasure 2 placed" << std::endl;
}

std::cout << "x: "; std::cin >> treasureLocations[4];
std::cout << "y: "; std::cin >> treasureLocations[5];
if (treasureLocations[0] && treasureLocations[1] == treasureLocations[4] && treasureLocations[5]) {
    std::cout << "You can't enter the same place twice! try again" << std::endl;
    setupGrid();
} else if (treasureLocations[2] && treasureLocations[3] == treasureLocations[4] && treasureLocations[5]) {
    std::cout << "You can't enter the same place twice! try again" << std::endl;
    setupGrid();
} else {
    std::cout << "Treasure 3 placed" << std::endl;
}

for (size_t i = 0; i < 6; i++) {
    if (treasureLocations[i] > 7) {
        std::cout << "You entered a number bigger then the size of the grid! try again.." << std::endl;
        setupGrid();
    }
}
```

Like I mentioned this code was put into a function so that if the user enters the same information twice I can easily reset the inputs and the first player wont have to reset the program completely and it wont crash.

```

tGrid[treasureLocations[0]][treasureLocations[1]] = true;
tGrid[treasureLocations[2]][treasureLocations[3]] = true;
tGrid[treasureLocations[4]][treasureLocations[5]] = true;

for (size_t x = 0; x < 8; x++) {
    for (size_t y = 0; y < 8; y++) {
        grid[x][y] = '-';
    }
}

```

After I know that the users information isn't going to cause the program to crash I set the background grid to the values that the first player entered keeping them for when player 2 enters information on their own turn, the last thing to do is to set the front-end grid to a dash this is going to be the default state of this grid.

The next step is to allow player 2 to enter a guess if they have more then 3 lives, this is another easy step that can be done with an if statement if the number of lives is greater then 0 then they can still make an attempted at finding the treasure.

```

if (lives > 0) {
    struct guessPos
    {
        int x;
        int y;
    };
    guessPos player2Guess;
    std::cout << "Player 2 Guess!" << std::endl;
    std::cout << "x: ";
    std::cin >> player2Guess.x;
    std::cout << "y: ";
    std::cin >> player2Guess.y;
}

```

Here is the code that checks the lives left as well as gets the seconds players input, to get the input I made a struct instead of recycling 2 ints or a small int array of 2.

```

if (player2Guess.x <= 7 && player2Guess.y <= 7) {
    if (tGrid[player2Guess.x][player2Guess.y] == true) {
        std::cout << "You found some treasure!" << std::endl;
        grid[player2Guess.x][player2Guess.y] = 'X';
        corretGuess++;
    }
    else {
        lives--;
        grid[player2Guess.x][player2Guess.y] = 'N';
    }
}

```

Next step is to make sure that the value entered are actually on the grid this is that first if statement the next line enters the second players guess into the background grid to see if that spot as the treasure and if it does it prints a line saying so and sets the display grid to an "X" marking that the player found treasure there and adds 1 to a int that's going to keep track of how many correct guess have been taken. If the guess is wrong, then the second player need to lose a life and then it need to be marked on the display grid by marking it with an "N".

```

else {
    std::cout << "You entered a number not on the grid! -1 life!" << std::endl;
    lives--;
}

displayGrid();
checkForCorretGuesses();
step();

```

If the second player does enter a number off the grid, then they lose a life and the game tells them so.

The last thing to do is to display the grid so that the players get updated on what is happening to the game, next the game is going to check to see if 2 correct guesses have been made and if not then the step function calls it's self so that the second player can take another go at guessing and if the second player is out of lives this code will run ending the game!

```

else {
    std::cout << "Player 1 wins!" << std::endl;
}

```

This is what the function for checking the amount of times the second player guessed correctly

```

void checkForCorretGuesses() {
    if (corretGuess == 3) {
        std::cout << "Player 2 wins!" << std::endl;
        system("pause");
        exit(0);
    }
}

```

It's a if statement that checks against the variable that holds the value of correct guesses and if its equal to 2 then it tells player 2 that they are the winner it will then pause the game until a player presses a key to exit the program.

Because of the way I wrote all my code, putting all the code that is going to be called more then once into a function so that it can be reused better my main function loops like this

```

int main()
{
    setupGrid();
    displayGrid();
    step();

    system("pause");
    return 0;
}

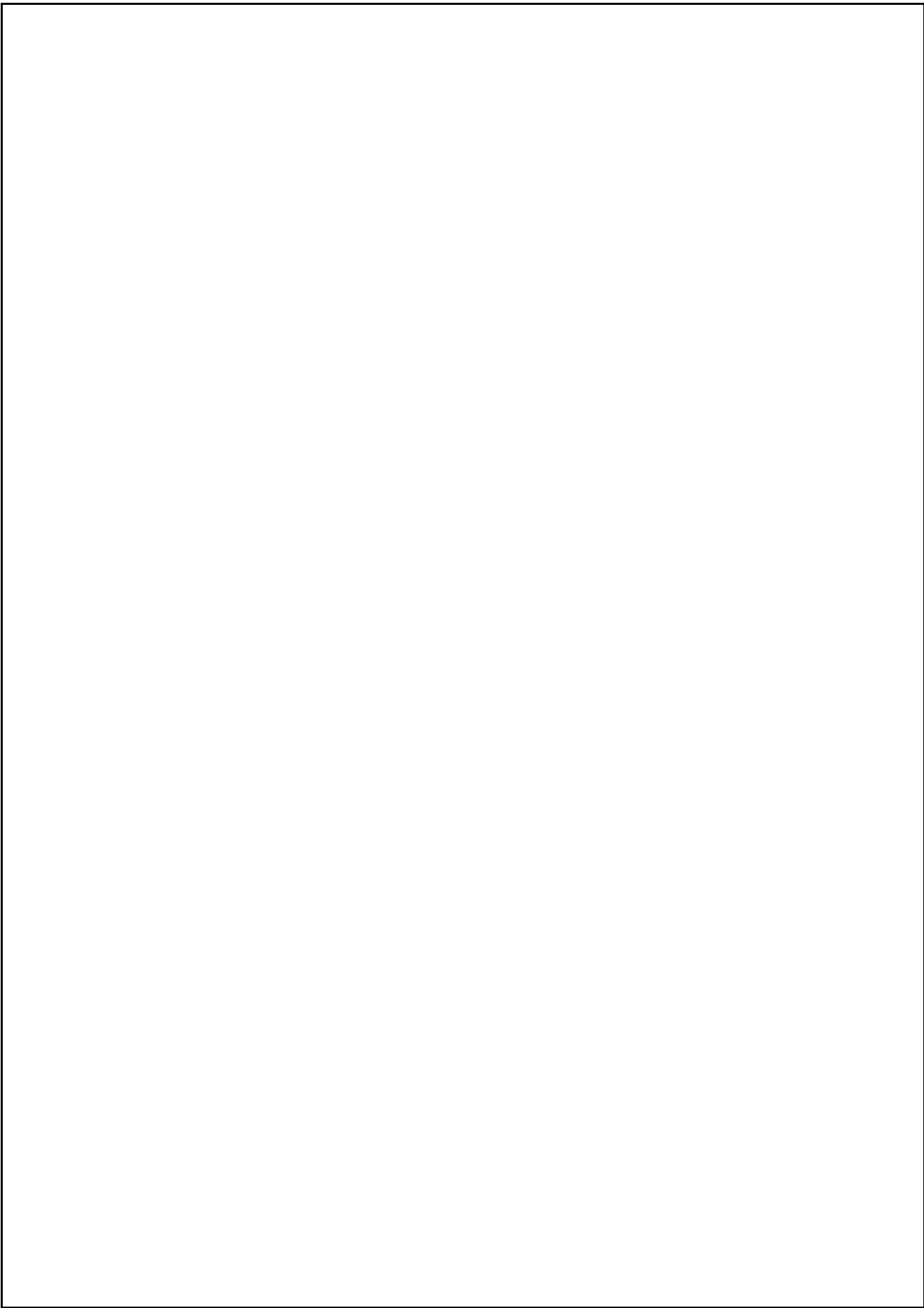
```

It sets up the game with the first function and displays the grid and then tries to step forward letting the second player take their turns, this could be pushed down to only 2 functions by putting displayGrid into the bottom of the setup function and then step and setup would be the only things left or they could be renamed to player one turn and player twos turn.

```
Player 1 place the treasure!
x: 1
y: 1
Treasure 1 placed
x: 2
y: 2
Treasure 2 placed
x: 3
y: 3
Treasure 3 placed
 0 1 2 3 4 5 6 7
0 - - - - - - -
1 - - - - - - -
2 - - - - - - -
3 - - - - - - -
4 - - - - - - -
5 - - - - - - -
6 - - - - - - -
7 - - - - - - -
Player 2 Guess!
x:
```

Here is a screenshot of the player ones turn followed by the table being displayed after player ones turn ends, next it will look like the screenshot below where player 2 will be taking their turn.

```
Player 2 Guess!
x: 2
y: 2
You found some treasure!
 0 1 2 3 4 5 6 7
0 - - - - - - -
1 - - - - N - -
2 - - X - - - -
3 - - - - - - -
4 - - - - - - -
5 - - - - - - -
6 - - - - - - -
7 - - - - - - -
Player 2 Guess!
x: 1
y: 7
 0 1 2 3 4 5 6 7
0 - - - - - - -
1 - - - - N - N
2 - - X - - - -
3 - - - - - - -
4 - - - - - - -
5 - - - - - - -
6 - - - - - - -
7 - - - - - - -
Player 2 Guess!
x:
```



FINAL GRADE

50/100

GENERAL COMMENTS

Instructor

Exercise 1 (50) 30

Exercise 2 (30) 20

Exercise 3 (20) 0

There is not much evidence of the problem solving strategy that you applied and you have not subdivided the problem to create appropriate functions.

Exercise 2 contains a lot of code but again it is not as well structured as we would like to see. There is some evidence of testing but I wanted a more formal/structured approach. I can see you are moving towards the solution.

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5



Comment 1

The placing of each treasure should be in a loop



Comment 2

The logic here seem flawed, what do you mean by `if(treasureLocations[0] &&`

PAGE 6

