COMP500 / ENSE501: Week 7 – Exercise:

EXERCISE NAME: *Treasure Finder*

Design (using pseudo code), implement (using C) and test (using Visual Studio Enterprise 2017) a simple program where the user moves around a 2-dimensional grid by moving north, east, south, or west.

The user must be prompted for their move as a single character input and be able to repeatedly make moves until they choose to quit. The virtual world is limited to a range from -10 to 10 inclusive on each axis.

When moving around the user may also encounter hidden treasure!

Declare and define at least three functions in addition to the **`main`** function. Do not use global variables or **`goto`**!

The program must start as follows with the user located at the (x, y) coordinate of (0, 0):

```
Find the hidden treasure!
-------------------------

Press 'Q' to quit at any time...

You are at (0, 0). Move (N/E/S/W)?
```

The user is then able to input their move choice or decide to quit. The program must support upper- and lower-case input.

Moving north and south changes the y-coordinate position. Moving east and west changes the x-coordinate position.

For example, moving north increases the y-coordinate by 1 as follows:

```
You are at (0, 0). Move (N/E/S/W)? n

You are at (0, 1). Move (N/E/S/W)?
```

Another example, moving south decreases the y-coordinate by 1 as follows:

```
You are at (4, 2). Move (N/E/S/W)? S

You are at (4, 1). Move (N/E/S/W)?
```

Another example, moving east increases the x-coordinate by 1 as follows:

```
You are at (7, -3). Move (N/E/S/W)? E

You are at (8, -3). Move (N/E/S/W)?
```

Another example, moving west decreases the x-coordinate by 1 as follows:

```
You are at (3, 9). Move (N/E/S/W)? w

You are at (2, 9). Move (N/E/S/W)?
```

When the user inputs an invalid command, the program must behave as follows:

```
You are at (4, 7). Move (N/E/S/W)? x
Invalid command!
```

Another example is as follows:

```
You are at (4, -4). Move (N/E/S/W)? 8
Invalid command!
```

When the user quits, the following message must be displayed:

```
You are at (5, 6). Move (N/E/S/W)? q

Goodbye!
```

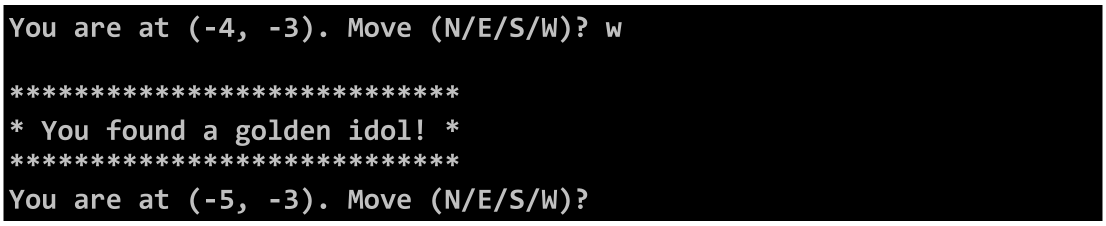
As the user moves around, they will encounter hidden treasure! The following hidden treasures must be discoverable by the player at the locations specified.

Hidden Treasure:	X Location:	Y Location:
Pirate’s chest	2	3
Golden idol	-5	-3
Precious gemstones	1	-2
Lost artwork	5	2
Your fifth treasure!	*You choose!*	*You choose!*

At (2, 3) there is a “pirate’s chest”:

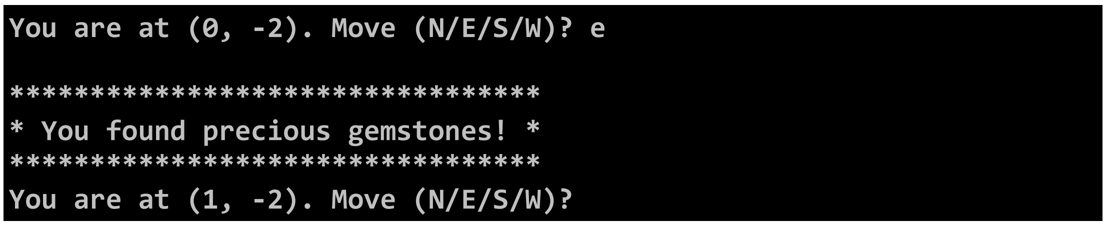
```
You are at (1, 3). Move (N/E/S/W)? e

********************************
* You found a pirate's chest! *
********************************
You are at (2, 3). Move (N/E/S/W)?
```

At (-5, -3) there is a “golden idol”:



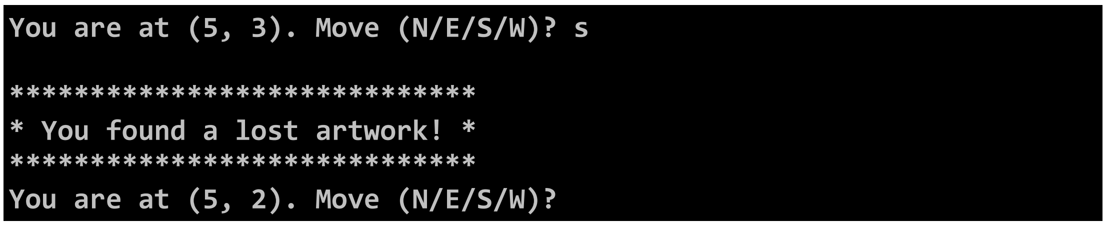
```
You are at (-4, -3). Move (N/E/S/W)? w

****************************
* You found a golden idol! *
****************************
You are at (-5, -3). Move (N/E/S/W)?
```

At (1, -2) there are “precious gemstones”:



```
You are at (0, -2). Move (N/E/S/W)? e

*********************************
* You found precious gemstones! *
*********************************
You are at (1, -2). Move (N/E/S/W)?
```

At (5, 2) there is a “lost artwork”:



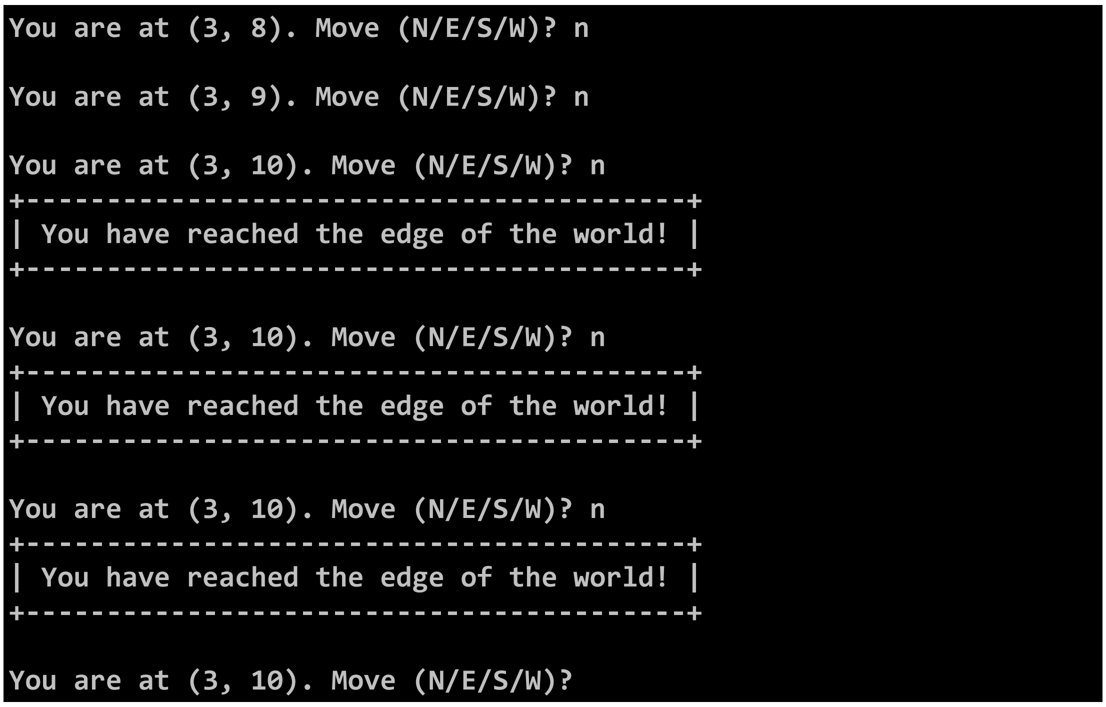
```
You are at (5, 3). Move (N/E/S/W)? s

*****************************
* You found a lost artwork! *
*****************************
You are at (5, 2). Move (N/E/S/W)?
```

Also, add a fifth treasure of your own design!

Ensure each treasure message is surrounded by * symbols as shown above.

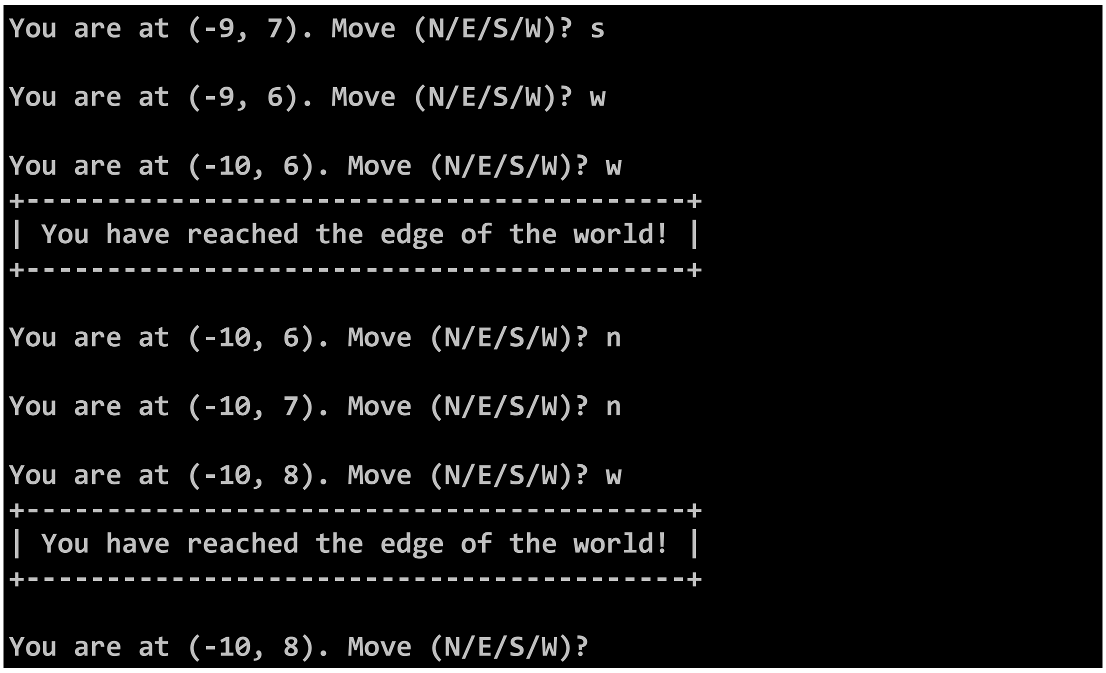
The virtual world also has a boundary. The user cannot move to a position where the x-coordinate or y-coordinate is less than -10 or greater than 10.

When the user tries to go beyond the boundaries of the world, the program must behave as follows:



```
You are at (3, 8). Move (N/E/S/W)? n

You are at (3, 9). Move (N/E/S/W)? n

You are at (3, 10). Move (N/E/S/W)? n
+------------------------------------------+
| You have reached the edge of the world! |
+------------------------------------------+

You are at (3, 10). Move (N/E/S/W)? n
+------------------------------------------+
| You have reached the edge of the world! |
+------------------------------------------+

You are at (3, 10). Move (N/E/S/W)? n
+------------------------------------------+
| You have reached the edge of the world! |
+------------------------------------------+

You are at (3, 10). Move (N/E/S/W)?
```

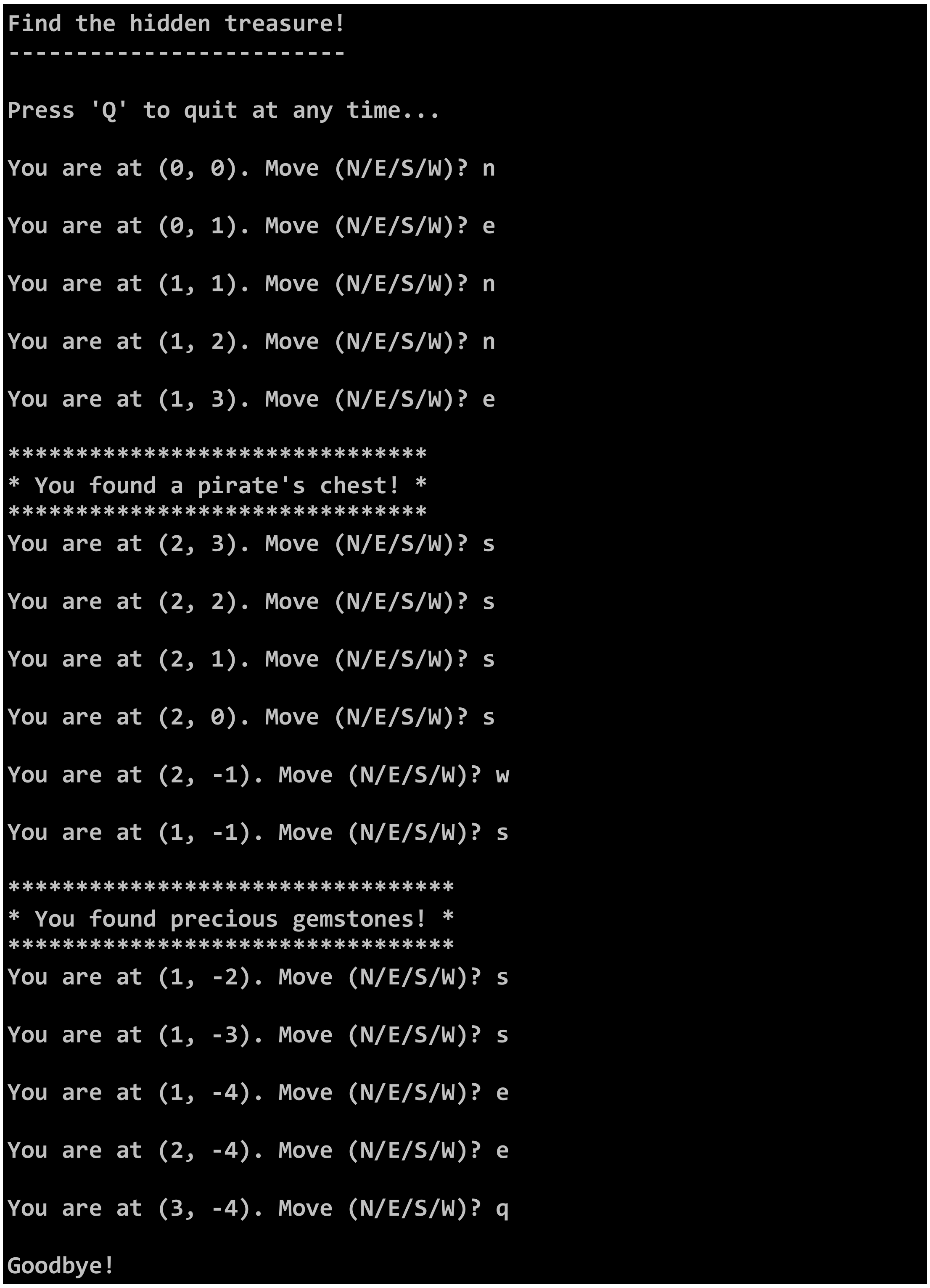
Another example is as follows:



```
You are at (-9, 7). Move (N/E/S/W)? s

You are at (-9, 6). Move (N/E/S/W)? w

You are at (-10, 6). Move (N/E/S/W)? w
+------------------------------------------+
| You have reached the edge of the world! |
+------------------------------------------+

You are at (-10, 6). Move (N/E/S/W)? n

You are at (-10, 7). Move (N/E/S/W)? n

You are at (-10, 8). Move (N/E/S/W)? w
+------------------------------------------+
| You have reached the edge of the world! |
+------------------------------------------+

You are at (-10, 8). Move (N/E/S/W)?
```

Ensure each boundary warning message is surrounded by an ASCII art box as shown above.

An example of the completed program, with user input:



```
Find the hidden treasure!
-------------------------

Press 'Q' to quit at any time...

You are at (0, 0). Move (N/E/S/W)? n

You are at (0, 1). Move (N/E/S/W)? e

You are at (1, 1). Move (N/E/S/W)? n

You are at (1, 2). Move (N/E/S/W)? n

You are at (1, 3). Move (N/E/S/W)? e

*******************************
* You found a pirate's chest! *
*******************************
You are at (2, 3). Move (N/E/S/W)? s

You are at (2, 2). Move (N/E/S/W)? s

You are at (2, 1). Move (N/E/S/W)? s

You are at (2, 0). Move (N/E/S/W)? s

You are at (2, -1). Move (N/E/S/W)? w

You are at (1, -1). Move (N/E/S/W)? s

*********************************
* You found precious gemstones! *
*********************************
You are at (1, -2). Move (N/E/S/W)? s

You are at (1, -3). Move (N/E/S/W)? s

You are at (1, -4). Move (N/E/S/W)? e

You are at (2, -4). Move (N/E/S/W)? e

You are at (3, -4). Move (N/E/S/W)? q

Goodbye!
```

Follow good programming standards for code layout whitespace, naming and commenting. Ensure your C source code can successfully compile. Test your program with a variety of input and ensure the resulting program output is as described above.