

Workshop #7

Introduction

In this workshop, you will code and execute a C language program applying user-defined data types (structures). You will be programming a small game that has hidden bombs and treasure along a path of variable distance. The game requires the player to enter move location commands to reveal what is hidden at a given position along the path. The object of the game is for the player to find as many treasures as possible before running out of moves or lives. Discovering a bomb will reduce the player's life count. Discovering a treasure will earn the player treasure points. Discovering both, a treasure with a bomb in the same location will reduce the player's life count and earn the player treasure points (consider it a life insurance payout). Prior to playing the game, the program will prompt the user to perform some upfront configurations to the player and the game components – these settings will define how the game is played.

Topic(s)

- [Structures](#)

Learning Outcomes

Upon successful completion of this workshop, you will have demonstrated the abilities to:

- to store data of different data types using a structure type
- to declare an object of structure type
- to access the members of an object of structure type
- To describe to your instructor what you have learned in completing this workshop

Part-1

Instructions

Download or clone workshop 7 (**WS07**) from <https://github.com/Seneca-144100/IPC-Workshops>

Note: If you use the download option, make sure you **EXTRACT** the files from the .zip archive file

Part-1 will focus on the **player** and **game configuration** settings in preparation for the gameplay which will be done in Part-2.

1. Carefully review the “[Part-1 Output Example](#)” (next section) to see how this program is expected to work (**Note:** This game is highly user-configurable and should be coded to implement the settings as defined by the user and not be limited to just the example provided – you will have to test your work thoroughly in both part's 1 and 2!)
2. Code your solution to Part-1 in the provided “**w7p1.c**” source code file.
3. You will need to create a user-defined data type called **PlayerInfo** which is used for configuring a player in the game with members that can store the following related information:
 - The **number of “lives”** a player can have for the game
 - A **character symbol** that will be used to represent the player
 - A counter to store the **number of “treasure’s”** found during the game
 - A **history of all past entered positions** entered by the player during the game (**HINT1:** you should size this array based on a macro that represents the **maximum path length** that a game can be configured for – see example output to see what the maximum is). **HINT2:** this should be an **int type** that stores only **1’s** and **0’s**).
4. You will need to create another user-defined data type called **GameInfo** which is used for configuring the game settings with members that can store the following related information:
 - The **maximum number of “moves”** a player can make for a game

- The **path length** (number of positions) the game path will have for a game
 - A series of **0's** and **1's** in an array that represents where **bombs** are buried along the path (hint: you should size this array based on a **macro that represents the maximum path length** that a game can be configured for – see example output to see what the maximum is)
 - A series of **0's** and **1's** in an array that represents where **treasure** is buried along the path (hint: you should size this array based on a **macro that represents the maximum path length** that a game can be configured for – see example output to see what the maximum is)
5. Configure the **player** (store these values to a variable of type **PlayerInfo**):
- Prompt to set the player's **character symbol** (any printable character that will represent the player)
 - Note: Place a **single space** before the % specifier in the scanf to properly read this value
`scanf(" %c"...`
 - Prompt to set the **number of lives** a player is limited to for the game
 - The value must be between **1** and **10** inclusive
 - Note: you should design your code so that the maximum value rule can be easily modified in one place, so you **do not need to make changes to the logic** of the program
 - Validation should repeat as many times as necessary until a valid value is entered
 - Make sure the history of moves (all user entered positions during gameplay) is set to a safe empty state – **you should assume there is potentially previous game data still stored that needs each element to be reset**
6. Configure the **game** (store these values to a variable of type **GameInfo**):
- Prompt to set the **length of the game path** (this is the number of positions in the path)
 - The value must be between **10** and **70**
 - The value must be a **multiple of 5**
 - Note: you should design your code so that these rules (values: 5, 10, 70) can be easily modified in one place, so you **do not need to make changes to the logic** of the program
 - Validation should repeat as many times as necessary until a valid value is entered
 - Prompt to set the **maximum number of moves** a player can make during gameplay
 - The value must be at least the value of the **player's "lives"** setting
 - The value cannot be greater than 75% of the game's **path length** setting (round down to nearest whole number)
 - Validation should repeat as many times as necessary until a valid value is entered
 - Prompt to set the **BOMB's** placements along the path (within the game's path length limits)
 - Values **must be entered 5 at a time** (sets of 5) until all positions along the set path length are set (space delimited)
 - Reminder: The multiple of 5 rule can be modified with another version of this application and should be coded with this mind (see note at the beginning of #6)
 - A '**1**' value represents a **hidden bomb**, while a '**0**' value represents **no bomb**
 - Note: You do not need to validate for **1's** and **0's**; you may assume this is entered properly
 - Prompt to set the **TREASURE** placements along the path (within the game's path length limits)
 - The same rules apply as described for the bomb settings
 - As the last major step, **display a summary** of the values entered that will define the gameplay

Part-1 Output Example (Note: Use the YELLOW highlighted user-input data for submission)

```
=====
Treasure Hunt!
=====
```

PLAYER Configuration

```
-----
Enter a single character to represent the player: @
Set the number of lives: 0
    Must be between 1 and 10!
Set the number of lives: 11
    Must be between 1 and 10!
Set the number of lives: 3
Player configuration set-up is complete
```

GAME Configuration

```
-----
Set the path length (a multiple of 5 between 10-70): 9
    Must be a multiple of 5 and between 10-70!!!
Set the path length (a multiple of 5 between 10-70): 71
    Must be a multiple of 5 and between 10-70!!!
Set the path length (a multiple of 5 between 10-70): 19
    Must be a multiple of 5 and between 10-70!!!
Set the path length (a multiple of 5 between 10-70): 35
Set the limit for number of moves allowed: 2
    Value must be between 3 and 26
Set the limit for number of moves allowed: 27
    Value must be between 3 and 26
Set the limit for number of moves allowed: 10
```

BOMB Placement

```
-----
Enter the bomb positions in sets of 5 where a value
of 1=BOMB, and 0=NO BOMB. Space-delimit your input.
(Example: 1 0 0 1 1) NOTE: there are 35 to set!
```

```
Positions [ 1- 5]: 0 0 0 0 1
Positions [ 6-10]: 1 0 0 1 1
Positions [11-15]: 1 0 1 1 1
Positions [16-20]: 0 1 0 0 0
Positions [21-25]: 1 0 1 0 0
Positions [26-30]: 0 0 0 1 0
Positions [31-35]: 1 0 1 0 1
```

BOMB placement set

TREASURE Placement

```
-----
Enter the treasure placements in sets of 5 where a value
of 1=TREASURE, and 0=NO TREASURE. Space-delimit your input.
(Example: 1 0 0 1 1) NOTE: there are 35 to set!
```

```
Positions [ 1- 5]: 0 0 1 0 0
Positions [ 6-10]: 1 1 1 0 1
Positions [11-15]: 1 1 0 1 0
Positions [16-20]: 0 1 0 0 0
```

Positions [21-25]: 1 1 0 1 0

Positions [26-30]: 1 0 1 0 0

Positions [31-35]: 0 1 1 1 1

TREASURE placement set

GAME configuration set-up is complete...

TREASURE HUNT Configuration Settings

Player:

Symbol : @

Lives : 3

Treasure : [ready for gameplay]

History : [ready for gameplay]

Game:

Path Length: 35

Bombs : 00001100111011101000101000001010101

Treasure : 00100111011101001000110101010001111

=====
~ Get ready to play TREASURE HUNT! ~
=====

Part-2

Instructions

Part-2 involves implementing the gameplay logic based on the configuration settings done from Part-1.

1. Review the “[Part-2 Output Example](#)” (next section) to see how the application is expected to work

Reminder:

This game is highly user-configurable and should be coded to implement the settings as defined by the user and not be limited to just the example provided – you will have to test your work thoroughly in both part’s 1 and 2!

2. Code your solution to Part-2 in the provided “**w7p2.c**” source code file.
3. Display the “board” which includes:
 - Line-1: The player’s most recent move position identified using the **player’s** set **symbol**
 - Line-2: The game path:
 - - : (hyphen) represents **undiscovered** locations
 - ! : represents **bombs**
 - \$: represents **treasure**
 - & : represents both a **bomb** and **treasure**
 - . : represents a visited location that had neither a **bomb** nor a **treasure**
 - Hint: Use the **player’s** history member array to determine if you should reveal the hidden bomb and/or treasure, if a player has visited the position/location, then check what symbol to show by checking the **game’s** bombs and treasure member array’s
 - Line-3: Position/location ruler (**major**) which will show the 1st number in every 10 positions
 - (10, 20, 30, etc.)
 - Line-4: Position/location ruler (**minor**) which shows each numeric position starting at 1

Note:

The ‘ruler’ helps the user quickly locate positions and identify previously entered move commands. Hint: This is dynamically displayed based on the **game’s** setting for the **path length** member

4. Display the **player’s** statistics.
 - Lives: is a counter of how many lives are remaining (when this is zero, gameplay ends)
 - Treasures: is a counter of how many treasures were found during gameplay
 - Moves Remaining: is a countdown of remaining moves (when this is zero, gameplay ends)
 - Use the following (add the variables for substitution accordingly where needed):

```
printf("+-----+\n");
printf("  Lives: %2d  | Treasures: %2d  | Moves Remaining: %2d\n"...
printf("+-----+\n");
```

5. Prompt for the **player’s** next **move** (location along the path)
 - The entered value must be at least 1 and no more than the **game’s** configuration setting for the **path length**
 - Validation should repeat as many times as necessary until a valid value is entered
6. Check to see if the entered location was **previously visited**
 - You should refer to the **player’s** **history** array to see if the location was previously visited (the value will be **1**)
 - If the location was previously visited, display a meaningful message to indicate the location has already been visited
 - Do NOT deduct a move from the **game’s** **move counter**
7. If the entered location was not previously visited:

- Record the location to the **player's history** array by setting the appropriate element value to 1 (the index is determined by the entered location)
 - Reduce the **moves counter** by 1
 - Check the **game's bomb** member array (the index is determined by the entered location) to see if there is a hidden bomb (value will be 1)
 - Reduce the **player's lives** counter by 1
 - Display an appropriate message (use symbol: [!] to denote a bomb)
 - Check the **game's treasure** member array (the index is determined by the entered location) to see if there is a hidden treasure (value will be 1)
 - Increase the **player's treasure** counter by 1
 - Display an appropriate message (use symbol: [\$] to denote a treasure)
 - Check for BOTH a **bomb** AND a **treasure**
 - Check both the **bomb** and **treasure** member arrays to see if a value of 1 is set for both at the same location
 - Update the player's counters accordingly (bomb: **reduce lives**, treasure: **increase treasure** counter)
 - Display an appropriate message (use symbol: [&] to denote a bomb AND treasure, the treasure is considered a "life insurance payout")
 - If there is no bomb or treasure at the location entered by the user
 - Display an appropriate message (use symbol: [.] to denote nothing found)
- Note: The symbols used in the messages will match to what is shown in the game's board when displayed (step #3 "line-2")

8. Keep iterating (looping) (from #3) until the gameplay ends based on the following criteria:

- The number of player **lives** reaches 0
- The number of allowed **moves** reaches 0

9. Display a "Game Over" message along with an exit/end of program message.

Part-2 Output Example (Note: Use the **YELLOW** highlighted user-input data for submission)

```
=====
      Treasure Hunt!
=====

PLAYER Configuration
-----
Enter a single character to represent the player: V
Set the number of lives: 0
      Must be between 1 and 10!
Set the number of lives: 11
      Must be between 1 and 10!
Set the number of lives: 3
Player configuration set-up is complete

GAME Configuration
-----
Set the path length (a multiple of 5 between 10-70): 9
      Must be a multiple of 5 and between 10-70!!!
Set the path length (a multiple of 5 between 10-70): 41
      Must be a multiple of 5 and between 10-70!!!
```

Set the path length (a multiple of 5 between 10-70): **19**
Must be a multiple of 5 and between 10-70!!!
Set the path length (a multiple of 5 between 10-70): **20**
Set the limit for number of moves allowed: **2**
Value must be between 3 and 15
Set the limit for number of moves allowed: **16**
Value must be between 3 and 15
Set the limit for number of moves allowed: **10**

BOMB Placement

Enter the bomb positions in sets of 5 where a value
of 1=BOMB, and 0=NO BOMB. Space-delimit your input.
(Example: 1 0 0 1 1) NOTE: there are 20 to set!

Positions [1- 5]: **1 0 0 1 1**
Positions [6-10]: **1 1 0 0 0**
Positions [11-15]: **0 0 1 1 1**
Positions [16-20]: **1 0 0 0 0**

BOMB placement set

TREASURE Placement

Enter the treasure placements in sets of 5 where a value
of 1=TREASURE, and 0=NO TREASURE. Space-delimit your input.
(Example: 1 0 0 1 1) NOTE: there are 20 to set!

Positions [1- 5]: **0 1 1 0 0**
Positions [6-10]: **0 0 0 0 0**
Positions [11-15]: **1 1 0 0 1**
Positions [16-20]: **0 1 1 1 1**

TREASURE placement set

GAME configuration set-up is complete...

TREASURE HUNT Configuration Settings

Player:

Symbol : V
Lives : 3
Treasure : [ready for gameplay]
History : [ready for gameplay]

Game:

Path Length: 20
Bombs : 10011110000011110000
Treasure : 01100000001100101111

=====
~ Get ready to play TREASURE HUNT! ~
=====

|||||||1|||||||2
12345678901234567890

```
+-----+
Lives:  3 | Treasures:  0 | Moves Remaining: 10
+-----+
Next Move [1-20]: 0
Out of Range!!!
Next Move [1-20]: 21
Out of Range!!!
Next Move [1-20]: 8

=====> [.] ...Nothing found here... [.]

      V
      .
-----
|||||||1|||||||2
12345678901234567890
+-----+
Lives:  3 | Treasures:  0 | Moves Remaining:  9
+-----+
Next Move [1-20]: 10

=====> [.] ...Nothing found here... [.]

      V
      .
-----
|||||||1|||||||2
12345678901234567890
+-----+
Lives:  3 | Treasures:  0 | Moves Remaining:  8
+-----+
Next Move [1-20]: 1

=====> [!] !!! BOOOOOM !!! [!]

      V
      !-----
      .
-----
|||||||1|||||||2
12345678901234567890
+-----+
Lives:  2 | Treasures:  0 | Moves Remaining:  7
+-----+
Next Move [1-20]: 15

=====> [&] !!! BOOOOOM !!! [&]
=====> [&] $$$ Life Insurance Payout!!! [&]

      V
      !-----&-----
      .
-----
|||||||1|||||||2
12345678901234567890
+-----+
Lives:  1 | Treasures:  1 | Moves Remaining:  6
+-----+
Next Move [1-20]: 20
```


=====> [\$] \$\$\$ Found Treasure! \$\$\$ [\$]

```

      V
!-----.-.-&---$
|||||||1|||||||2
12345678901234567890
+-----+
Lives:  1  | Treasures:  2  | Moves Remaining:  5
+-----+
Next Move [1-20]: 8
```

=====> Dope! You've been here before!

```

      V
!-----.-.-&---$
|||||||1|||||||2
12345678901234567890
+-----+
Lives:  1  | Treasures:  2  | Moves Remaining:  5
+-----+
Next Move [1-20]: 3
```

=====> [\$] \$\$\$ Found Treasure! \$\$\$ [\$]

```

      V
!-$----.-.-&---$
|||||||1|||||||2
12345678901234567890
+-----+
Lives:  1  | Treasures:  3  | Moves Remaining:  4
+-----+
Next Move [1-20]: 5
```

=====> [!] !!! BOOOOOM !!! [!]

No more LIVES remaining!

```

      V
!-$-!--.-.-&---$
|||||||1|||||||2
12345678901234567890
+-----+
Lives:  0  | Treasures:  3  | Moves Remaining:  3
+-----+
```

```
#####
#   Game over!   #
#####
```

You should play again and try to beat your score!