Week 2: Day 004.1 – C++: Using References:

Create a Visual Studio Solution and Project for Day 004. Implement a function called **`RollTwoDice`** which can be called with two **`int`** parameters. The function must use pass-by-reference behaviour to send the result of rolling two different dice to the calling function. Do not change the return type from **`void`**!

The skeleton of this program is provided below:

```
/* #include required headers here */

void RollTwoDice(/* Insert your code here */);

int main()
{
    // Declare two local integers, representing the two dice.

    // Set each dice to hold a value of zero.

    // Print out the contents of the two dice after the call.

    // Call rollTwoDice with the local dice variables.

    // Print out the contents of the two dice after the call.

    return 0;
}

void RollTwoDice(/* Insert your code here */)
{
    /* Insert your code here */
}
```

Week 2: Day 004.2 – C++: Using Structures:

Add a new project to your Day 004 Visual Studio solution. Declare a structure which represents a Player in a 2D game environment. The Player must have fields for their position, a value for their health, and a Boolean to represent whether or not they are alive.

The skeleton of this program is provided below:

```
/* #include required headers here */

struct Player
{
    /* Insert your code here */
};

int main()
{
    return 0;
}
```

Next, write a function called `SetupPlayer()` which takes in a reference to a Player structure, and populates the fields of the structure with the following values: a position of (0, 0), a health of 100, and an alive state of `true`. Ensure you have a prototype and a definition for this function.

Next, write a function called `PrintPlayerDetails()` which takes in a `const` reference to a Player structure. In this function print out the player's details in the following format:

```
Player's Current State:
 -  Position: (0, 0)
 -  Health: 100
 -  Alive: Yes
```

Next, in the main function, add two local structure variables called `player1` and `player2`. Call `PrintPlayerDetails` with `player1` and then `player2`. Then call `SetupPlayer` with `player1` and `player2`. Then call `PrintPlayerDetails` again. The following is an example of this:

```
/* #include required headers here */

struct Player
{
    /* Insert your code here */
};

/* Insert Prototypes here */

int main()
{
    // Declare two local structure variables.

    // Call PrintPlayerDetails with player1.
    // Call PrintPlayerDetails with player2.

    // Call SetupPlayer with player1.
    // Call SetupPlayer with player2.

    // Call PrintPlayerDetails with player1.
    // Call PrintPlayerDetails with player2.

    return 0;
}

/* Insert function definitions here */
```

Explain the difference between the details printed before the calls to `SetupPlayer`.

Week 2: Day 004.3 – C++: Using Pointers and the Freestore:

Add a new project to your Day 004 Visual Studio solution. Declare a structure which represents an NPC (Non-Player Character) in a 2D game. Add fields to this structure to represent the follow details: strength, health, tiredness, position and whether the NPC is alive or not.

Next declare a function called **`CreateNPC()`** which takes no parameters and returns a pointer to an NPC structure.

In the **`CreateNPC`** function, allocate a NPC structure on the Freestore. Assign the fields of the NPC random values based upon the following criteria:

- 3 < Strength < 20
- 50 < Health < 200
- 1 < Tiredness < 15
- 0 < Position X < 100
- 0 < Position Y < 100
- Alive is true

Then return this allocation to the calling function.

Write another function called **`PrintNPCDetails()`** which takes in a NPC pointer parameter, and prints the details of the NPC in the following format:

```
NPC's Current State:
 -  Position: (0, 0)
 -  Health: 112
 -  Strength: 15
 -  Tiredness: 10
 -  Alive: Yes
```

In the **`main`** function, create an array of 10 NPC pointers.

Set each pointer in the array to be null using a loop.

Next, call **`CreateNPC`** ten times and store the resulting allocations in the NPC pointer array.

Finally, call **`PrintNPCDetails`** ten times, each time with address of an NPC stored on the Freestore.

The following is an example of this:

```
/* #include required headers here */

struct NPC
{
    /* Insert your code here */
};

/* Insert Prototypes here */

int main()
{
    // Declare NPC pointer array.

    // Call CreateNPC ten times, store the results.

    // Call PrintNPCDetails for each NPC in the array.

    return 0;
}

/* Insert function definitions here */
```

Before the program finishes, deallocate the Freestore allocations, and hence avoid a memory leak occurring!