Background: In this exercise, we will write a simplified form of the dice game Yahtzee. In this version, the goal is to get 3 matching dice, and if you can't do that, then you hope to at least get 2 matching dice.

We will represent a hand of 3 dice as a single three digit integer. So the hand 4-3-2 will be represented by the integer `432`.

The game is played like so:

1. Roll 3 dice.
2. If you do not have 3 matching dice:
    - If you have 2 matching dice (a pair), keep the pair and roll one die to replace the third die.
    - Otherwise, if you have no matching dice, keep the highest die and roll two dice to replace the other two dice.
3. Repeat step 2 one more time.
4. Finally, compute your score like so:
    - If you have 3 matching dice, your score is 20 + the sum of the matching dice. So if you have 4-4-4, your score is 20+4+4+4, or 32.
    - If you only have 2 matching dice, your score is 10 + the sum of the matching dice. So if you have 4-4-3, your score is 10+4+4, or 18.
    - If you have no matching dice, your score is the highest die.

With this in mind, write the function `playThreeDiceYahtzee(dice)` that takes an integer `dice` that represents all the rolls for a game of 3-Dice Yahtzee and returns both the resulting hand and the score for that hand.

Important Hints:

Since this is a large task, we will break it up into smaller, more manageable pieces. So we'll first write some helper functions that do part of the work, and then these helper functions will make our final function much easier to write. Be sure to write your functions in the same order as given here, since later functions will make use of earlier ones.

1. `handToDice(hand)`

    Write the function `handToDice(hand)` that takes a three digit integer `hand` and returns each of the three dice values in the hand. For example:

```
assert(handToDice(123) == (1,2,3))
assert(handToDice(214) == (2,1,4))
assert(handToDice(422) == (4,2,2))
```

Hint: You might find `//` and `%` useful here, and also `getKthDigit(n, k)`.

2. `diceToOrderedHand(a, b, c)`

   Write the function `diceToOrderedHand(a,b,c)` that takes three dice and returns them in a hand, which is a three digit integer. However, even if the dice are unordered, the resulting hand must be ordered so that the largest die is on the left and the smallest die is on the right. For example:

```
assert(diceToOrderedHand(1,2,3) == 321)
assert(diceToOrderedHand(6,5,4) == 654)
assert(diceToOrderedHand(1,4,2) == 421)
```

   Hint: You can use `max(a,b,c)` to find the largest of three values, and `min(a,b,c)` to find the smallest.

3. `playStep2(hand, dice)`

   Write the function `playStep2(hand, dice)` that plays step 2 of Yahtzee as explained above. This function takes `hand`, which is a three digit integer, and it also takes `dice`, which is an integer containing all the future rolls of the dice. For example, if `dice` is `5341`, then the next roll will be a 1, then the roll after that will be a 4, then a 3, and finally a 5. (Note that in a more realistic version of this game, instead of hard-coding the dice in this way, we'd probably use a random number generator.)

   The function should play step 2 with the given hand, using the given dice to get the next rolls as needed. At the end, the function returns the new hand, ordered, and the resulting dice that no longer contain the rolls that were just used. For example:

```
assert(playStep2(413, 2312) == (421, 23))
```

Here, the `hand` is `413`, and the future dice rolls are `2312`. What happens? Well, there are no matching dice in `413`, so we keep the highest die, which is a 4, and we replace the 1 and the 3 with new rolls. Since new rolls come from the right of our `dice`, our rolls are 2 and 1. So the new hand is `421`. Finally, the `dice` was `2312`, but we used two digits, so now it's just `23`. We return both the `hand` and the `dice`, so we return `(421, 23)`.

Here are some more examples. Be sure you carefully understand them:

```
assert(playStep2(413, 2345) == (544, 23))
assert(playStep2(544, 23) == (443, 2))
assert(playStep2(544, 456) == (644, 45))
```

Hints:

- You may wish to use `handToDice(hand)` at the start to convert the hand into the three individual dice.
- Then, you may wish to use `diceToOrderedHand(a,b,c)` at the end to convert the three dice back into a sorted hand.
- Also, remember to use `%` to get the one's digit, and to use `//=` to get rid of the one's digit.

4. `score(hand)`

Now write the function `score(hand)` that takes a three digit integer `hand`, and returns the score for that hand as explained above in step 4 of Yahtzee. For example:

```
assert(score(432) == 4)
assert(score(532) == 5)
assert(score(443) == 10+4+4)
assert(score(633) == 10+3+3)
assert(score(333) == 20+3+3+3)
assert(score(555) == 20+5+5+5)
```

Hint: The structure of this function is actually quite similar to the previous function.

5. `playThreeDiceYahtzee(dice)`

   This function takes one value, the dice with all the rolls for a game of 3-Dice Yahtzee. The function plays the game -- it does step 1 and gets the first three dice (from the right), then it does step 2 twice (by calling `playStep2(hand, dice)`, which you already wrote), and then it computes the score (by calling `score(hand)`, which you already wrote). The function should return two values -- the resulting hand and the score for that hand. For example:

```
assert(playThreeDiceYahtzee(2312413) == (432, 4))
assert(playThreeDiceYahtzee(2315413) == (532, 5))
assert(playThreeDiceYahtzee(2345413) == (443, 18))
assert(playThreeDiceYahtzee(2633413) == (633, 16))
assert(playThreeDiceYahtzee(2333413) == (333, 29))
assert(playThreeDiceYahtzee(2333555) == (555, 35))
```

   Hint: Note that `playStep2` takes the current hand and dice, but then it also returns the updated hand and dice. You should use parallel assignment to update the hand and dice with the result of the call to playStep2. That is, you should do something like this:

```
hand, dice = playStep2(hand, dice)
```