Problem 1: Basics

<table>
<thead>
<tr>
<th>Problem</th>
<th>True</th>
<th>False</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rust's Borrowing mechanism helps prevent double frees</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>There exist memory safe programs which Rust will not compile</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>All Rust expressions are Rust statements, but not all statements are expressions</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>Returning a ref (eg &amp;str) is always safe in rust operations</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>The lifetime of a piece a data is always the same as the scope of the variable associated with it</td>
<td>T</td>
<td>F</td>
</tr>
</tbody>
</table>

Problem 2: Ownership

Consider the following Rust Program

```rust
fn add2(n: i32) -> i32 {
    let mut n2 = n;
    n2 += 2;
    return n2;
}

fn put2(s: String) -> String {
    let mut s2 = s.clone();
    s2.push_str("_2");
    return s2;
}

fn main() {
    let a = 5;
    let b = add2(a);
    println!("a: {} b: {}", a, b);
    // -- no error from line 14 --
    let t = String::from("one");
    let u = put2(t);
    println!("t: {} u: {}", t, u);
    // -- ERROR from line 18 --
}
```

(A) Why will the compiler indicate there is an error at line 18 after the call to `put2()` on line 17?

The value of `t` gets moved because it is of type `String` which does not implement the copy trait. So at line 18, `t` no longer owns "one".

(B) `add2()` is very similar to `put2()` but after its call on line 13, there is no compile error on line 14. Why not? What is different between these functions?

`add2()` takes in a variable of type `i32` which has a copy trait.

(C) In the function `put2()`, will the data associated with parameter `s` be dropped by the end of the function or does the lifetime of that data last beyond the end of `put2()`? Justify your answer with a sentence.

The data associated with `s` will be dropped by the end of the function because of the transfer of ownership.

(D) How would you change the parameter type for `put2()` and its call at line 17 to "fix" the compiler problem?

The type of the parameter for `put2()` can be changed to `&String` and accordingly the function call as well. Alternatively you can call `t.clone()` when the variable is being passed in.
Consider the `to_binstring` function from project 7. We want you to do the same thing but return a hex number. Like in the project, we recommend using a data structure like a vector. Below, you may find some helpful Rust syntax. Additionally, you may use the provided global array of hex values to obtain the appropriate hex value you are looking for.

**CONSTRAINTS:** You may not use a print formatter nor can you use the built-in `to_hex()` function.

**POSSIBLY USEFUL BUILT-IN FUNCTIONS:** It is **NOT** necessary to use all of these in your solution.

```rust
vec.push(ele); // Pushes the element 'ele' | iter.rev(); // reverses an iterators direction
   // to end of the vector 'vec' |
string.push_str(&str); // appends the str | iter.next(); // returns an Option of the next
   // to string | // item in the iterator.
vec.len() // length of vector | option.unwrap(); // returns the item in an Option or
string.len // length of String | // panics if None
vec.iter(); // returns an iterator for vec | string.chars() // returns an iterator of chars
   | // over the a string
```

**EXAMPLES** of `to_hexstring()`:

```
// to_hexstring( 0) -> "0" | to_hexstring( 2) -> "2" | to_hexstring( 10) -> "A"
// to_hexstring( 32) -> "20" | to_hexstring(510) -> "1FE" | to_hexstring(1024) -> "400"
```

// a useful array whose elements may be indexed via HEX_ARR[i]
static HEX_ARR : [&str;16] =
   ["0","1","2","3","4","5","6","7","8","9","A","B","C","D","E","F"];

```rust
pub fn to_hexstring(num: usize) -> String {
    if num == 0 { // special case of 0
        return String::from("0");
    }

    let mut cur = num;
    let mut digits = vec![];
    while cur > 0 {
        let rem = cur % 16; // pushing in remainder of 16 each
        digits.push(HEX_ARR[rem]); // iteration, builds digits in
        cur = cur / 16; // reverse order
    }

    // // Reverse digits using a loop
    let mut rdigits = String::new();
    let len = digits.len();
    for i in 0..len {
        rdigits.push_str(digits[len-i-1]);
    }
    return rdigits;
}
```

OR

// Reverse digits using an iterator
let mut rdigits = String::new();
for d in digits.iter().rev() {
    rdigits.push_str(d);

```
return rdigits;

// Alternative reversals exist including versions that use a
// combination of ...rev().collect::<String>()
}