Problem 1: Basics

Rust’s Borrowing mechanism helps prevent double frees  

True False

There exist memory safe programs which Rust will not compile

True False

All Rust expressions are Rust statements, but not all statements are expressions

True False

Returning a ref (eg &str) is always safe in rust operations

True False

The lifetime of a piece a data is always the same as the scope of the variable associated with it

True False

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?

(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?

(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.

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

**CONCEPTS:** 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' to end of the vector 'vec'
string.push_str(&str); // appends the str to string
vec.len() // length of vector
string.len // length of String
vec.iter(); // returns an iterator for vec
string.chars() // returns an iterator of chars over the a string
```

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

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

```rust
static HEX_ARR : [&str;16] =
["0","1","2","3","4","5","6","7","8","9","A","B","C","D","E","F"];

pub fn to_hexstring(num: usize) -> String {
    //YOUR CODE HERE
}