ECE 421 Assignment 2

Jiannan Lu 1577618

1.

```
struct Bag<T> {
    items: [T; 3],
}
```

2.

```
fn bag_size<T>(bag: &Bag<T>) -> usize {
    std::mem::size_of_val(bag)
}
```

3.

```
struct Bag_u8 {
 items: [u8; 3],
}

struct Bag_u32 {
 items: [u32; 3],
}

fn bag_size_u8(bag: &Bag_u8) -> usize {
    std::mem::size_of_val(bag)
}

fn bag_size_u32(bag: &Bag_u32) -> usize {
    std::mem::size_of_val(bag)
}

fn main() {
    let b1 = Bag_u8 {items: [1u8, 2u8, 3u8], };
    let b2 = Bag_u32 {items: [1u32, 2u32, 3u32], };

    println!("size of First Bag = {} bytes", bag_size_u8(&b1));
    println!("size of Second Bag = {} bytes", bag_size_u32(&b2));
}
```

4.

```
/* output:
size of vec1_iter = 16 bytes
size of vec_chained = 32 bytes
size of vec_flattened = 48 bytes
*/

fn main() {
    let vec1 = vec![12, 32, 13];
    let vec2 = vec![44, 55, 16];
```

```rust
    {
        let vec1_iter = vec1.iter();
        println!("size of vec1_iter = {} bytes",
std::mem::size_of_val(&vec1_iter));
    }
    {
        let vec_chained = vec1.iter().chain(vec2.iter());
        println!("size of vec_chained = {} bytes",
std::mem::size_of_val(&vec_chained));
    }
    {
        let vec1_2=vec![vec1, vec2];
        let vec_flattened = vec1_2.iter().flatten();
        println!("size of vec_flattened = {} bytes",
std::mem::size_of_val(&vec_flattened));
    }
}
```

5.

```rust
/* output:
size of vec1_iter = 8 bytes
size of vec_chained = 8 bytes
size of vec_flattened = 8 bytes
*/

fn main() {
    let vec1 = Box::new(vec![12, 32, 13]);
    let vec2 = Box::new(vec![44, 55, 16]);
    {
        let vec1_iter = Box::new((*vec1).iter());
        println!("size of vec1_iter = {} bytes",
std::mem::size_of_val(&vec1_iter));
    }
    {
        let vec_chained =
Box::new((*vec1).iter().chain((*vec2).iter()));
        println!("size of vec_chained = {} bytes",
std::mem::size_of_val(&vec_chained));
    }
    {
        let vec1_2=Box::new(vec![*vec1, *vec2]);
        let vec_flattened = Box::new((*vec1_2).iter().flatten());
        println!("size of vec_flattened = {} bytes",
std::mem::size_of_val(&vec_flattened));
    }
}
```

6.
In q4, the iterator is a variable (pointer) stored on the stack, which point to the original vector stored on the heap. The size of them is obtained from memory allocated for ‘the actual original vector’s values’.
In q5, the iterator is now stored on the heap with allocated memory from Box instead, having a pointer (box) point to that data.
7.
Polymorphism is a concept that allows objects of different types to be treated as objects of a common base type.
Yes, rust supports the polymorphism, ``Rust instead uses generics to abstract over different possible types and trait bounds to impose constraints on what those types must provide. This is sometimes called bounded parametric polymorphism.''
8.
2 time. On line 206 and 216, equal is being called by “call   example::equal”
9.
With -O flag, equal is being called by 0 time. It’s likely that the compiler recognizes that the two value are not the same and simplified the equal operation by replacing equal(x,y) to False, and equal(x.len(), y.len()) to True.