A recursive type has infinite size because it doesn't have an indirection.

Erroneous code example:

```
struct ListNode {
    head: u8,
    tail: Option<ListNode>, // error: no indirection here so impossible to
                            //        compute the type's size
}
```

When defining a recursive struct or enum, any use of the type being defined from inside the definition must occur behind a pointer (like `Box`, `&` or `Rc`). This is because structs and enums must have a well-defined size, and without the pointer, the size of the type would need to be unbounded.

In the example, the type cannot have a well-defined size, because it needs to be arbitrarily large (since we would be able to nest `ListNode`s to any depth). Specifically,

```
size of `ListNode` = 1 byte for `head`
                   + 1 byte for the discriminant of the `Option`
                   + size of `ListNode`
```

One way to fix this is by wrapping `ListNode` in a `Box`, like so:

```
struct ListNode {
    head: u8,
    tail: Option<Box<ListNode>>,
}
```

This works because `Box` is a pointer, so its size is well-known.