1 Introduction to Binary Operations

Consider the following s-expression: `(sub1 (sub1 (add1 73)))`. Looking at the code discussed in the lecture handout, and assuming `main` runs, what does the stack and heap look like when

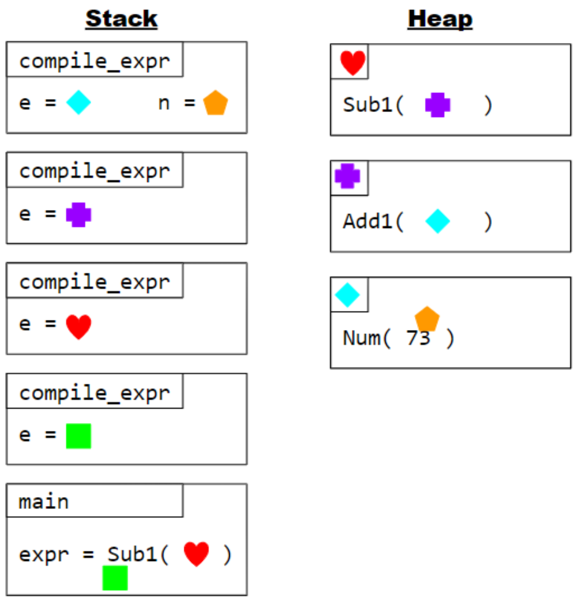
```
format!("mov rax, {}", *n)
```

evaluates?

We'll take a look at the function calls of `compile_expr(&expr)`. First, note that Rust will store objects on the stack unless you allocate it on the heap. Recall, from the previous lecture, we have the AST representation

```
Expr::Sub1(
    Box::new(Expr::Sub1(
        Box::new(Expr::Add1(
            Box::new(
                Expr::Num(73)
            )
        ))
    ))
)
```

Our code initially calls `compile_expr(&expr)`, where `&expr` is a reference to the above object. Note that the outer `Expr::Sub1` is in the stack, but the data in each of the `Enums` will be allocated in the heap. In any case, after calling the function initially, it makes a recursive call with the argument being the held data of the inner object. This repeats until we reach the end (when we have the `Num`).


Stack
Heap
compile_expr
e =
n =
compile_expr
e =
compile_expr
e =
compile_expr
e =
main
expr = Sub1( )
Sub1( )
Add1( )
Num(73)