

```
enum Expr {
    Num(i32),
    True, False,
    Add1(Box<Expr>),
    Plus(Box<Expr>, Box<Expr>),
    Let(String, Box<Expr>, Box<Expr>),
    Id(String),
    Eq(Box<Expr>, Box<Expr>),
    If(Box<Expr>, Box<Expr>, Box<Expr>)
}

fn new_label(l : &mut i32, s : &str) ->
    let current = *l;
    *l += 1;
    format!("{}_{current}")
}
```

```
#[link(name = "our_code")]
extern "C" {
    #[link_name = "\x01our_code_starts_here"]
    fn our_code_starts_here() -> i64;
}

fn print_stack(i : i64) {
    if(i == 1) { println!("false"); }
    else if(i == 3) { println!("true"); }
    else { println!("{} 3", i > 71) }
}

fn main() {
    let i : i64 = unsafe { our_code_starts_here() };
    println!("{}", i); print_stack(i)
}
```

```

fn compile_expr(e: &Expr, si: i32, env: &HashMap<String, i32>, l: &mut i32) -> String {
    match e {
        Expr::Num(n) => format!("mov rax, {}", *n << 1),
        Expr::True => format!("mov rax, {}", 1),
        Expr::False => format!("mov rax, {}", 0),
        Expr::Eq(e1, e2) => {
            let e1_result = compile_expr(e1, si, env, l);
            let e2_offset = compile_expr(e2, si + 1, env, l);
            format!("
                cmp rax, [rsp - {}]
                jne .L1
                mov rax, 1
                .L1: mov rax, 3
            ", e1_result, e2_offset)
        }
        Expr::If(cond, thn, els) => {
            let end_label = new_label(l, "ifend");
            let else_label = new_label(l, "ifelse");
            let cond_instrs = compile_expr(cond, si, env, l);
            let thn_instrs = compile_expr(thn, si, env, l);
            let els_instrs = compile_expr(els, si, env, l);
            format!(
                "{}\n                cmp rax, 1\n                je {}\n                {}\n                {}\n                {}\n                {}\n            ",
                cond_instrs,
                else_label,
                thn_instrs,
                else_label,
                els_instrs,
                end_label
            )
        }
    }
}

```

e1 result
si + B
[rsp - offset]

mov rax, 1
cmov rax, 3

TODO:
 - Fix the bugs!
 - Implement Eq, equal values and

error here if rax is non-boolean
 need to jump to the end!

TODO:

- Fix the bugs!
- Implement Eq, which evaluates to true for equal values and false otherwise

```
→ cat test/if-true.snek  
(if true 500 7)  
→ make test/if-true.run  
→ cat test/if-true.s
```

```
section .text
global our_code_starts_here
our_code_starts_here:
    mov rax, 3
    cmp rax, 1
    je ifelse_1
        mov rax, 1000
ifelse_1:
    mov rax, 14
ifend_0:
ret
→ ./test/if-true.run
```

```
→ cat test/if-false.snek
(if false 500 7)
→ make test/if-false.run
make: `test/if-false.run' is up to date.
→ cat test/if-false.s
```

```
section .text
global our_code_starts_here
our_code_starts_here:
    mov rax, 1
    cmp rax, 1
    je ifelse_1
        mov rax, 1000
ifelse_1:
    mov rax, 14
ifend_0:
ret
→ ./test/if-false.run
14
```

What about command-line input and reporting dynamic errors?

What an error should look like?

```
→ cat test/eq-diff.snek  
(= 1 true)  
→ ./test/eq-diff.run  
An error occurred 1
```

```
→ cat test/input-as-num.snek  
(+ input 7)  
→ ./test/input-as-num.run 11  
18
```

```
fn compile_expr(e: &Expr, si: i32, env: &HashMap<String, i32>, l: &mut i32) -> String {  
    match e {  
        Expr::Num(n) => format!("mov rax, {}", *n << 1),  
        Expr::True => format!("mov rax, {}", 3),  
        Expr::False => format!("mov rax, {}", 1),  
  
        Expr::Id(s) if s == "input" =>  
            "mov rax, rdi"  
  
        Expr::Eq(e1, e2) => {  
            let e1_instrs = compile_expr(e1, si, env, l);  
            let e2_instrs = compile_expr(e2, si + 1, env, l);  
            let offset = si * 8;  
            format!()  
                "  
                {e1_instrs}  
                mov [rsp - {offset}], rax  
                {e2_instrs}  
                mov rbx, rax  
                xor rbx, [rsp - {offset}]  
                test rbx, 1  
                jne throw_error  
                cmp rax, [rsp - {offset}]  
                mov rbx, 3  
                mov rax, 1  
                cmov rax, rbx  
                "  
        }  
    }  
}
```

```
fn main() -> std::io::Result<()> {  
    ... as before ...  
    let mut labels = 0;  
    let result = compile_expr(&expr, 2, &HashMap::new(), &mut labels);  
    let asm_program = format!()  
        "  
        section .text  
        global our_code_starts_here  
        extern snek_error  
        throw_error:  
            mov rdi, _____  
            jmp snek_error  
        our_code_starts_here:  
            {}  
            ret  
        ",  
        result  
    );
```

```
fn parse_arg(v : &Vec<String>) -> i64 {  
    if v.len() < 2 { return 1 }  
    let s = &v[0];  
    if s == "true" { 3 }  
    else if s == "false" { 1 }  
    else { s.parse::<i64>().unwrap() << 1 }  
}  
fn snek_error(err_code: i64) {  
    eprintln!( "An error: {} ", err_code )  
    exit(1);  
}  
fn main() {  
    let args: Vec<String> = env::args().collect();  
    let input = parse_arg(&args);  
    let i : i64 = unsafe { our_code_starts_here(input) };  
    print_value(i);  
}
```