src/main.rs

```
use std::mem;
use std::fs::File;
use std::env;
use std::io::prelude::*;
use sexp::*;
use sexp::Atom::*;

enum Expr {
  Num(i32),
  Add1(Box<Expr>),
  Sub1(Box<Expr>)
}
```

UPDATED AST

```
fn parse_expr(s : &Sexp) -> Expr {
  match s {
    Sexp::Atom(I(n)) =>
      Expr::Num(i32::try_from(*n).unwrap()),
    Sexp::List(vec) =>
      match &vec[..] {
        [Sexp::Atom(S(op)), e] if op == "add1" =>
          Expr::Add1(Box::new(parse_expr(e))),
        [Sexp::Atom(S(op)), e] if op == "sub1" =>
          Expr::Sub1(Box::new(parse_expr(e))),
        _ => panic!("parse error")
       },
    _ => panic!("parse error")
  }
}

fn compile_expr(e : &Expr) -> String {
  match e {
       Expr::Num(n) => format!("mov rax, {}", *n),
       Expr::Add1(subexpr) =>
        compile_expr(subexpr) + "\nadd rax, 1",
       Expr::Sub1(subexpr) =>
        compile_expr(subexpr) + "\nsub rax, 1"
  }
}

fn main() -> std::io::Result<()> {
  let args: Vec<String> = env::args().collect();

  let in_name = &args[1];
  let out_name = &args[2];

  let mut in_file = File::open(in_name)?;
  let mut in_contents = String::new();
  in_file.read_to_string(&mut in_contents)?;

  let expr = parse_expr(&parse(&in_contents).unwrap());
  let result = compile_expr(&expr);
  let asm_program = format!("
section .text
global our_code_starts_here
our_code_starts_here:
  {}
  ret
", result);

  let mut out_file = File::create(out_name)?;
  out_file.write_all(asm_program.as_bytes())?;

  Ok(())
}
```

Makefile

```
test/%.s: test/%.snek src/main.rs
    cargo run -- $< test/$*.s

test/%.run: test/%.s runtime/start.rs
    nasm -f elf64 test/$*.s -o runtime/our_code.o
    ar rcs runtime/libour_code.a runtime/our_code.o
    rustc -L runtime/ runtime/start.rs -o test/$*.run
```

runtime/start.rs

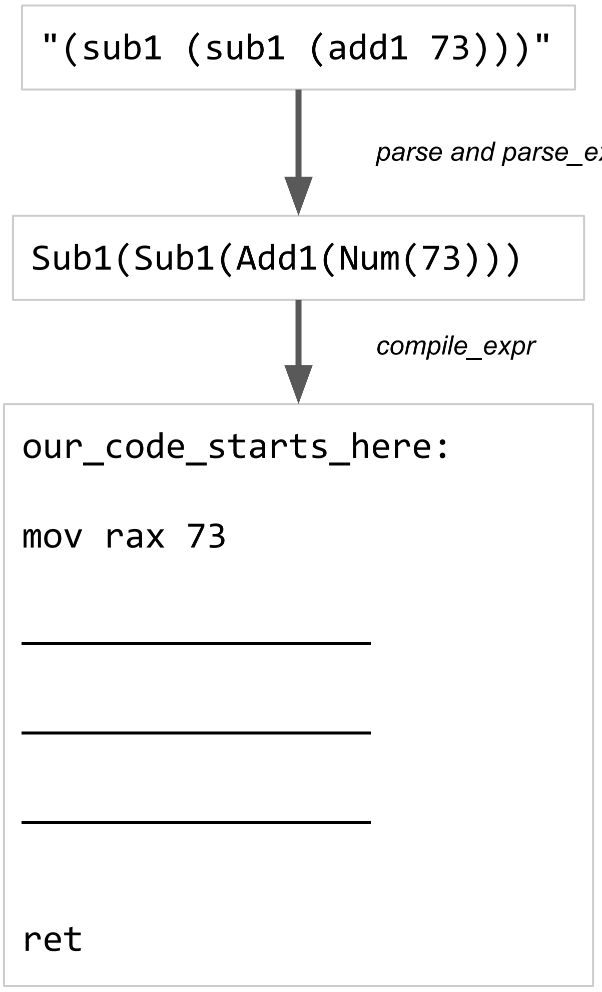
```
#[link(name = "our_code")]
extern "C" {
  fn our_code_starts_here() -> i64;
}

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

test/add.snek

```
(sub1 (sub1 (add1 73)))
```

```
$ make test/add.run
```



```
"(sub1 (sub1 (add1 73)))"
```

parse and parse_expr

```
Sub1(Sub1(Add1(Num(73)))
```

compile_expr

```
our_code_starts_here:

mov rax 73

______________

______________

______________

ret
```

High level language

Nano

Int ① ADDER

Bool ②

If-then else

Loop

Func

Tuples, lists, trees

HOFs / Closures

Garbage Collection

Type Check + OPTI

$\xrightarrow{CSE231}$ X86

Adder

```
e ::= n
    | add1 (e)
    | sub1 (e)
```

"Grammar"

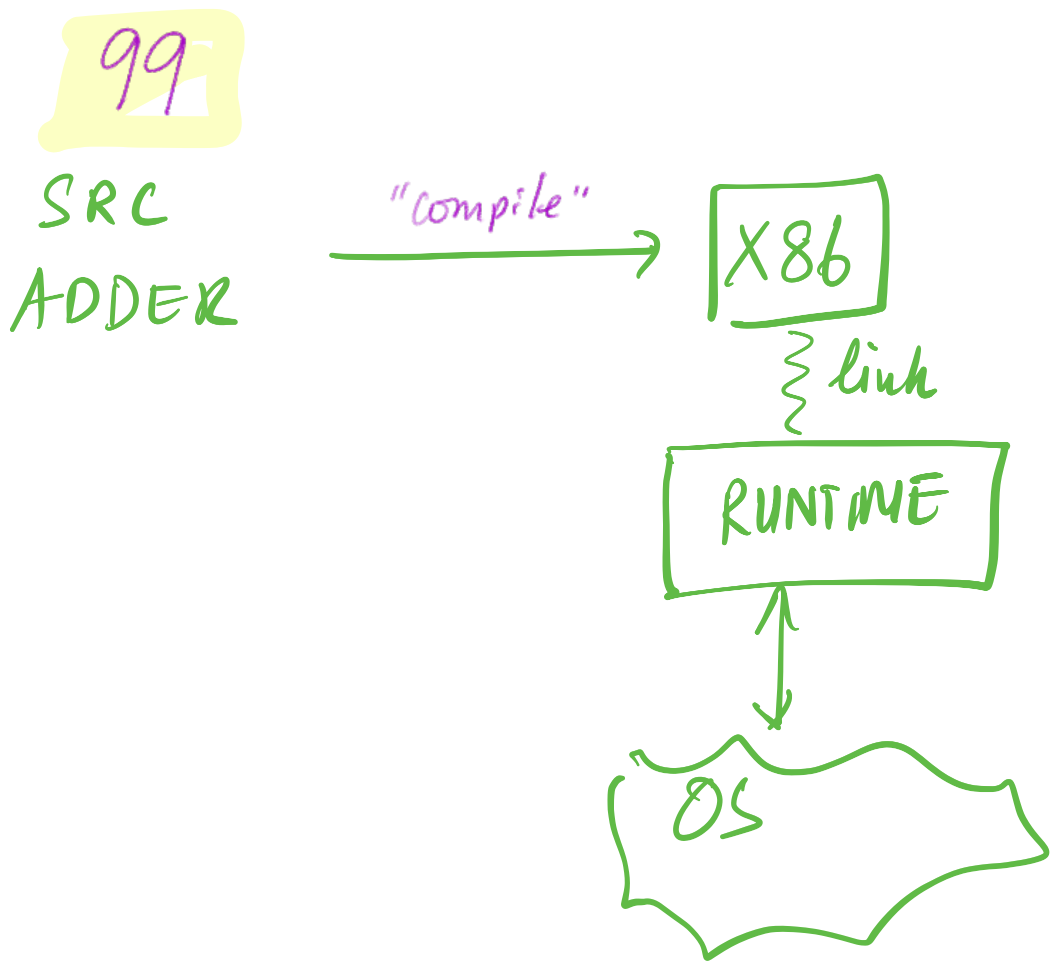
99

$\texttt{add1(99)} \longrightarrow 100$

$\texttt{add1(}\underbrace{\texttt{add1(42)}}\texttt{)} \longrightarrow 44$

$\texttt{sub1(add1(add1(42)))} \longrightarrow 43$

"Tests"



RJ's Feature Checklist

1 "think of feature" eg add1, sub1

2 write tests eg

$$\boxed{\begin{array}{l}\texttt{(add1 99)}\\ \texttt{(add1 (sub1 99))}\end{array}}$$

3 update AST

4 parser : String $\rightarrow$ AST

"(add1 99)"
"(add1 (add1 99))"
"99"

5 compiler :: AST $\longrightarrow$ Vec<X86>

```
99                  →    mov rax, 99

(add1 99)           →    mov rax, 99
                         add rax, 1

(add1 (add1 99))    →    mov rax, 99
                         add rax, 1
                         add rax, 1
```