

Let's add **local variables** and **binary ops** to our compiler

```
expr := <number>
      | (add1 <expr>)
      | (let (<name> <expr>) <expr>)
      | <name>
      | (+ <expr> <expr>)
```

Result

Programs

Stack Layout

Assembly

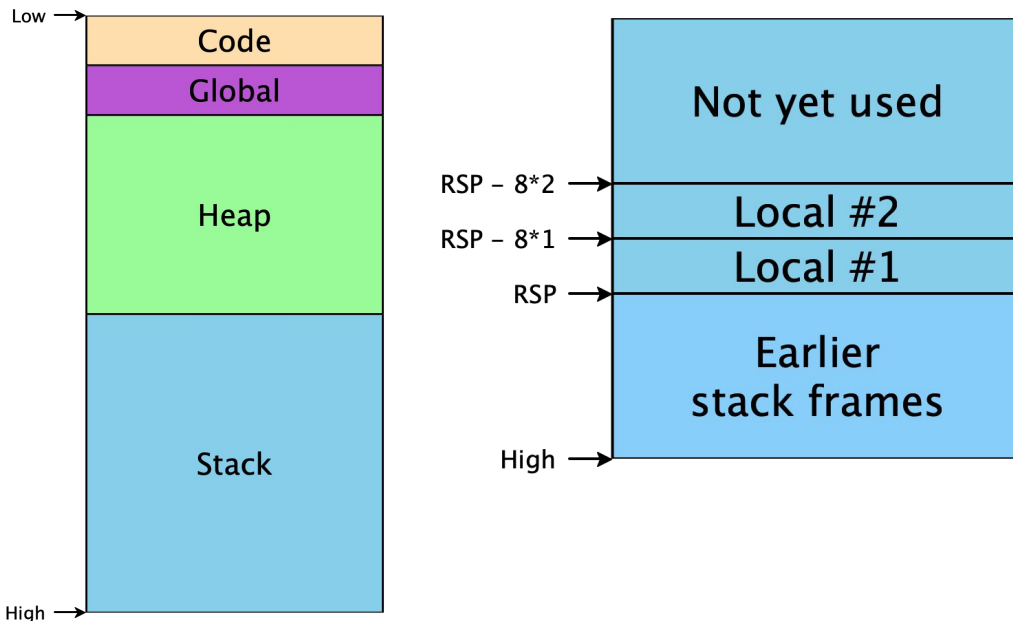
```
(let (x 10)
  (add1 x))
```

```
(let (x 10)
  (let (y (add1 x))
    (let (z (add1 y))
      (add1 z))))
```

```
(let (y (let (x10)
  (add1 x)))
  (add1 y))
```

```
(let (x 10)
  (let (x (add1 x))
    (add1 x)))
```

```
(let (a 1)
  (let (c
    (let (b (add1 a))
      add1(b)))
    add1 c))
```



Let's add **local variables** and **binary ops** to our compiler

(+ (100 50) 2)

```
expr := <number>
      | (add1 <expr>)
      | (let (<name> <expr>) <expr>)
      | <name>
      | (+ <expr> <expr>)
```

What assembly is produced?

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

```
enum Val {
  Reg(Reg),
  Imm(i32),
}
```

```
enum Reg {
  RAX,
}
```

```
enum Instr {
  IMov(Val, Val),
  IAdd(Val, Val),
  ISub(Val, Val),
  IMul(Val, Val),
}
```

```
fn compile(e: &Expr, is: &mut Vec<Instr>){
  match e {
    Expr::Num(n) => {
      is.push(is.push(mov(reg(Reg::RAX), imm(*n))))
    }
    Expr::Add1(e1) => {
      compile(e1, is);
      is.push(add(reg(Reg::RAX), imm(1)))
    }
  }
}
```

Rust Immutable Data Structures: <https://docs.rs/im/latest/im/>

Module `im::hashmap`

An unordered map.

An immutable hash map using [hash array mapped tries](#).

Most operations on this map are $O(\log_x n)$ for a suitably high x that it should be nearly $O(1)$ for most maps. Because of this, it's a great choice for a generic map as long as you don't mind that keys will need to implement [Hash](#) and [Eq](#).

pub fn `update(&self, k: K, v: V) -> Self`

Construct a new hash map by inserting a key/value mapping into a map.

If the map already has a mapping for the given key, the previous value is overwritten.

Time: $O(\log n)$

Examples

```
let map = hashmap!{};
assert_eq!(
    map.update(123, "123"),
    hashmap!{123 => "123"} );
```

pub fn `get<BK>(&self, key: &BK) -> Option<&V>` where

BK: [Hash](#) + [Eq](#) + [?Sized](#),

K: [Borrow](#)<BK>,

Get the value for a key from a hash map.

Time: $O(\log n)$

Examples

```
let map = hashmap!{123 => "lol"};
assert_eq!( map.get(&123), Some(&"lol") );
```