```
type expr =
  | ENum of int
  | EBool of bool
  | EId of string
  | EIf of expr * expr * expr
  | ELet of string * expr * expr
  | EPlus of expr * expr
  | EApp of string * expr
type def =
  | Def of string * string * expr
type prog =
  | Prog of def list
```

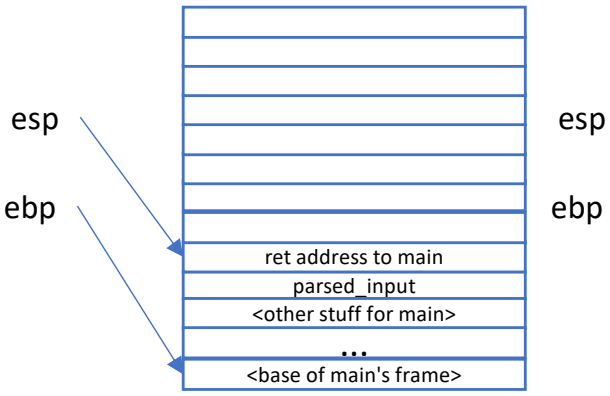
```
(def (f x)
  (if x
    (+ x (f (+ x -1)))
    0))
(def (our_main input)
  (f input))
```

```
Prog([
  Def("f", "x",
   EIf(EId("x"),
       EPlus(Eid("x"),
             EApp("f", EPlus(EId("x"), ENum(-1)))),
       ENum(0)));
  Def("our_main", "input",
   EApp("f", EId("input")))
])
```

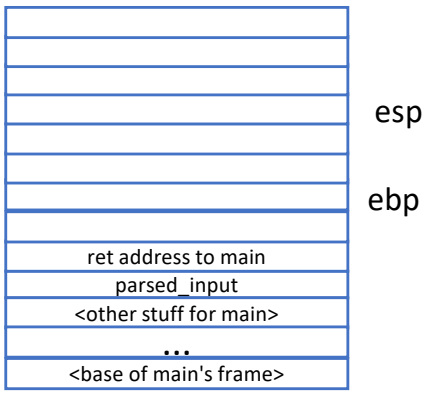
```
int main(int argc, char** argv) {
  int parsed_input = 0;
  if(argc > 1) { parsed_input = atoi(argv[1]); }
  int result = our_code_starts_here(parsed_input);
  printf("%d\n", result);
  fflush(stdout);
  return 0;
}
```

Assume true = 1, false = 0 (no tagging of values).
EIf takes the false branch on 0, true branch otherwise.

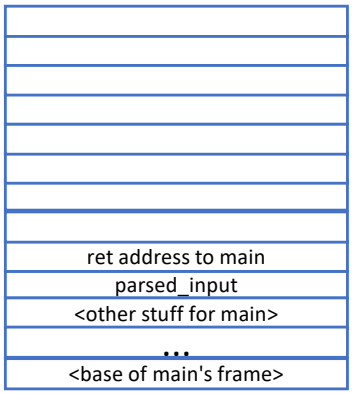
Stack at our_code_starts_here



Stack at our_main (you fill)



Stack at f (first time, you fill)




```
let rec stack_depth e =
```

```
let compile_def (d : def) =
  match d with
    | Def(name, arg, body) ->
      let depth = stack_depth body in
      let bodyis = e_to_is body 2 [(arg, 1)] in
      [
        sprintf "%s:" name;
        sprintf "sub esp, %d" (depth * 4);
      ]
      @ bodyis @
      [
        sprintf "mov esp, ebp";
        "ret"
      ]
let rec e_to_is e si env =
  match e with
    | EApp(name, arg) ->
      let after_label = gen_tmp "after_call" in
      let argis = e_to_is arg si env in
      argis @
      [
        "push ebp";
        sprintf "push %s" after_label;
        "mov ebp, esp";
        "push eax";
        sprintf "jmp %s" name;
        sprintf "%s:" after_label;
        "pop ebp";
      ]
```

f:

```
sub esp, 20
mov eax, [ebp - 4]
cmp eax, 0
je else2
mov eax, [ebp - 4]
mov [ebp - 8], eax
mov eax, [ebp - 4]
mov [ebp - 12], eax
mov eax, -1
mov [ebp - 16], eax
mov eax, [ebp - 12]
add eax, [ebp - 16]
_____
_____
_____
_____
jmp f
after_call3:
pop ebp
mov [ebp - 12], eax
mov eax, [ebp - 8]
add eax, [ebp - 12]
jmp after_if1
else2:
mov eax, 0
after_if1:
mov esp, ebp
ret
```

our_main:

```
sub esp, 4
mov eax, [ebp - 4]
push ebp
push after_call4
mov ebp, esp
push eax
jmp f
after_call4:
pop ebp
mov esp, ebp
ret
```

our_code_starts_here:

```
mov eax, [esp+4]
push ebp
push after_main
mov ebp, esp
push eax
jmp our_main
after_main:
pop ebp
ret
```