TASK: Add booleans, comparisons, and if to this language

What are your motivating examples?

A C-style reminder

```
if(0) answer = 5;
else answer = 2;
```

cmp compares a register to a value (or other register) and sets status bits for jumps.

`je` is "jump if equal". There are `jg` (jump greater), `jle` (jump less than or equal), and more, `jne` (jump not equal)

```
mov rax, 0
cmp rax, 0
je else_branch
mov rax, 5
jmp after_if
else_branch:
mov rax, 2
after_if:
```

```
(* Any changes to grammar and expr? *)

(*
expr := <number>
     |  (<op> <expr>)
     |  (let (<name> <expr>) <expr>)
     |  (+ <expr> <expr>)

op   := inc | dec
*)

type op =
  | Inc
  | Dec

type expr =
  | ENum of int
  | EOp of op * expr
  | EId of string
  | ELet of string * expr * expr
  | EPlus of expr * expr
```

```
let int_of_string_opt s =
  try
    Some(int_of_string s)
  with
    Failure _ -> None

let rec sexp_to_expr (se : Sexp.t) : expr =
  match se with
    (* Which cases need to change?
       Which new cases are needed? *)
```

```
let parse (s : string) : expr =
  sexp_to_expr (Sexp.of_string s)
```

```
(* Compiles a source program string to an x86 string *)
let compile (program : string) : string =
  let ast = parse program in
  let instrs = e_to_is ast 1 [] in
  let instrs_str = (String.concat "\n" instrs) in
  sprintf "
section .text
global our_code_starts_here
our_code_starts_here:
  %s
  ret\n" instrs_str

(* Any changes to compile? *)
```

```
open Printf

let stackloc i = (i * 8)
let stackval i = sprintf "[rsp - %d]" (stackloc i)
type tenv = (string * int) list

let rec find (env : tenv) (x : string) : int option =
  match env with
    | [] -> None
    | (y, i)::rest ->
      if y = x then Some(i) else find rest x

let rec e_to_is (e : expr) (si : int) (env : tenv) =
  match e with
    (* Which cases need to change?
       Which new cases are needed? *)
```

```
#include <stdio.h>

extern int our_code_starts_here()
asm("our_code_starts_here");

int main(int argc, char** argv) {
  int result = our_code_starts_here();
  printf("%d\n", result);
  return 0;
}

(* Any changes to main? *)
```

63-bit, 2's complement number

tag bit (1=num, 0=bool)

Value	**Representation (bits)**	**Representation**	
		hex	**decimal**
9	0000 0000 0000 ... 0000 0000 000**1 0011**	0x000...00**13**	19
-2	1111 1111 1111 ... 1111 1111 1111 **1101**	0xFFF...FFF**D**	-3
33	0000 0000 0000 ... 0000 0000 __________	0x000...00__	___
true	**1**111 1111 1111 ... 1111 1111 1111 111**0**	0x**F**FF...FFF**E**	-2
false	**0**111 1111 1111 ... 1111 1111 1111 111**0**	0x**7**FF...FFF**E**	

true/false

62 bits for future value representations!

```
let rec e_to_is (e : expr) si env =
  match e with
    | ENum(n) ->


    | EBool(b) (* b is true or false *) ->
```

TASK: Add booleans, comparisons, and if to this language

What are your motivating examples?

```
(* Any changes to grammar and expr? *)

(*
expr := <number>
     |  (<op> <expr>)
     |  (let (<name> <expr>) <expr>)
     |  (+ <expr> <expr>)

op   := inc | dec
*)

type op =
  | Inc
  | Dec

type expr =
  | ENum of int
  | EOp of op * expr
  | EId of string
  | ELet of string * expr * expr
  | EPlus of expr * expr
```

```
let int_of_string_opt s =
  try
    Some(int_of_string s)
  with
    Failure _ -> None

let rec sexp_to_expr (se : Sexp.t) : expr =
  match se with
    (* Which cases need to change?
       Which new cases are needed? *)
```

```
let parse (s : string) : expr =
  sexp_to_expr (Sexp.of_string s)
```

```
(* Compiles a source program string to an x86 string *)
let compile (program : string) : string =
  let ast = parse program in
  let instrs = e_to_is ast 1 [] in
  let instrs_str = (String.concat "\n" instrs) in
  sprintf "
section .text
global our_code_starts_here
our_code_starts_here:
  %s
  ret\n" instrs_str

(* Any changes to compile? *)
```

A C-style reminder

```
if(0) answer = 5;
else answer = 2;
```

cmp compares a register to a value (or other register) and sets status bits for jumps.

je is "jump if equal". There are jg (jump greater), jle (jump less than or equal), and more, jne (jump not equal)

```
mov rax, 0
cmp rax, 0
je else_branch
mov rax, 5
jmp after_if
else_branch:
mov rax, 2
after_if:
```

```
open Printf

let stackloc i = (i * 8)
let stackval i = sprintf "[rsp - %d]" (stackloc i)
type tenv = (string * int) list

let rec find (env : tenv) (x : string) : int option =
  match env with
    | [] -> None
    | (y, i)::rest ->
      if y = x then Some(i) else find rest x

let rec e_to_is (e : expr) (si : int) (env : tenv) =
  match e with
    (* Which cases need to change?
       Which new cases are needed? *)
```

```
#include <stdio.h>

extern int our_code_starts_here()
asm("our_code_starts_here");

int main(int argc, char** argv) {
  int result = our_code_starts_here();
  printf("%d\n", result);
  return 0;
}

(* Any changes to main? *)
```

TASK: Add booleans, comparisons, and if to this language

What are your motivating examples?

```
(* Any changes to grammar and expr? *)

(*
expr := <number>
     |  (<op> <expr>)
     |  (let (<name> <expr>) <expr>)
     |  (+ <expr> <expr>)

op   := inc | dec
*)

type op =
  | Inc
  | Dec

type expr =
  | ENum of int
  | EOp of op * expr
  | EId of string
  | ELet of string * expr * expr
  | EPlus of expr * expr
```

```
let int_of_string_opt s =
  try
    Some(int_of_string s)
  with
    Failure _ -> None

let rec sexp_to_expr (se : Sexp.t) : expr =
  match se with
    (* Which cases need to change?
       Which new cases are needed? *)
```

```
let parse (s : string) : expr =
  sexp_to_expr (Sexp.of_string s)
```

```
(* Compiles a source program string to an x86 string *)
let compile (program : string) : string =
  let ast = parse program in
  let instrs = e_to_is ast 1 [] in
  let instrs_str = (String.concat "\n" instrs) in
  sprintf "
section .text
global our_code_starts_here
our_code_starts_here:
  %s
  ret\n" instrs_str

(* Any changes to compile? *)
```

A C-style reminder

```
if(0) answer = 5;
else answer = 2;
```

cmp compares a register to a value (or other register) and sets status bits for jumps.

je is "jump if equal". There are jg (jump greater), jle (jump less than or equal), and more, jne (jump not equal)

```
mov rax, 0
cmp rax, 0
je else_branch
mov rax, 5
jmp after_if
else_branch:
mov rax, 2
after_if:
```

```
open Printf

let stackloc i = (i * 8)
let stackval i = sprintf "[rsp - %d]" (stackloc i)
type tenv = (string * int) list

let rec find (env : tenv) (x : string) : int option =
  match env with
    | [] -> None
    | (y, i)::rest ->
      if y = x then Some(i) else find rest x

let rec e_to_is (e : expr) (si : int) (env : tenv) =
  match e with
    (* Which cases need to change?
       Which new cases are needed? *)
```

```
#include <stdio.h>

extern int our_code_starts_here()
asm("our_code_starts_here");

int main(int argc, char** argv) {
  int result = our_code_starts_here();
  printf("%d\n", result);
  return 0;
}

(* Any changes to main? *)
```