```
(*
expr := <number>
     |  (let (<name> <expr>) <expr>)
     |  (+ <expr> <expr>)
     |  <name>
*)
type expr =
  | ENum of int
  | EId of string
  | ELet of string * expr * expr
  | EPlus of expr * expr
```

```
let stackloc i = (i * 4)
let stackval i = sprintf "[esp - %d]" (stackloc i)
type tenv = (string * int) list

(* Assume si starts at 1 in the first call *)
let rec e_to_is (e : expr) (si : int) (env : tenv) =
  match e with
    | EPlus(e1, e2) ->
```

```
let e1is = e_to_is e1 si env in
let e2is = e_to_is e2 si env in
e1is @
["mov ebx, eax"] @
e2is @
["add eax, ebx"]
```

```
let e1is = e_to_is e1 si env in
let e2is = e_to_is e2 (si + 1) env in
e1is @
[sprintf "mov %s, eax" (stackval si)] @
e2is @
[sprintf "mov %s, eax" (stackval (si + 1));
 sprintf "mov eax, %s" (stackval si);
 sprintf "add eax, %s" (stackval (si + 1))]
```

(+ 1 2)

```
let e1is = e_to_is e1 si env in
let e2is = e_to_is e2 si env in
e1is @
["mov ebx, eax"] @
e2is @
["add eax, ebx"]
```

```
let e1is = e_to_is e1 si env in
let e2is = e_to_is e2 (si + 1) env in
e1is @
[sprintf "mov %s, eax" (stackval si)] @
e2is @
[sprintf "mov %s, eax" (stackval (si + 1));
 sprintf "mov eax, %s" (stackval si);
 sprintf "add eax, %s" (stackval (si + 1))]
```

(+ 5 (+ 1 3))

```
let e1is = e_to_is e1 si env in
let e2is = e_to_is e2 si env in
e1is @
["mov ebx, eax"] @
e2is @
["add eax, ebx"]
```

```
let e1is = e_to_is e1 si env in
let e2is = e_to_is e2 (si + 1) env in
e1is @
[sprintf "mov %s, eax" (stackval si)] @
e2is @
[sprintf "mov %s, eax" (stackval (si + 1));
 sprintf "mov eax, %s" (stackval si);
 sprintf "add eax, %s" (stackval (si + 1))]
```

(+ 5 (+ 1 3))

```
(*
expr := <number>
     |  (let (<name> <expr>) <expr>)
     |  (+ <expr> <expr>)
     |  <name>
*)
type expr =
  | ENum of int
  | EId of string
  | ELet of string * expr * expr
  | EPlus of expr * expr
```

```
let stackloc i = (i * 4)
let stackval i = sprintf "[esp - %d]" (stackloc i)
type tenv = (string * int) list

(* Assume si starts at 1 in the first call *)
let rec e_to_is (e : expr) (si : int) (env : tenv) =
  match e with
    | EPlus(e1, e2) ->
```

```
let e1is = e_to_is e1 si env in
let e2is = e_to_is e2 si env in
e1is @
["mov ebx, eax"] @
e2is @
["add eax, ebx"]
```

```
let e1is = e_to_is e1 si env in
let e2is = e_to_is e2 (si + 1) env in
e1is @
[sprintf "mov %s, eax" (stackval si)] @
e2is @
[sprintf "mov %s, eax" (stackval (si + 1));
 sprintf "mov eax, %s" (stackval si);
 sprintf "add eax, %s" (stackval (si + 1))]
```

(+ 1 2)

```
(let (x (let (y 10) (inc y)))
  (dec x))
```

A. 9
B. 10
C. 11
D. 12
E. Error

```
(let (x (let (x 10) (inc x)))
  (dec x))
```

A. 9
B. 10
C. 11
D. 12
E. Error

```
(let (x (let (x 10) (inc x)))
  (dec x))
```

A. 9
B. 10
C. 11
D. 12
E. Error