ANF : Administrative Normal Form

$(1+2)-(3-4)$

```
let x = 1+2 in
let y = 3-4 in
x - y
```

```
let x = 4 in
(4+2)-(3+x)
```

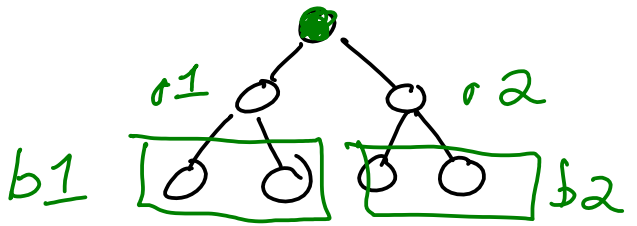
user
```
let x = 4 in
```

gen
```
let x = 4+2 in
let y = 3+x in
x - y
```

need fresh names

```
let rec anf_inner (e:expr) : ((string * c_expr) list * c_expr)
  match e with
  | EInt(n) -> ([], IInt n)
  |  ⋮
  | EPlus(e1, e2) ->
      let (b1, c1) = anf_inner e1 in
      let (b2, c2) = anf_inner e2 in
      let r1 = ("x", c1) in
      let r2 = ("y", c2) in
      (b1@[r1]@b2@[r2], CPlus(IVar "x", IVar "y"))
```

ghfklji



i (f(g(), h()), j(k(), 10))

(4+3) + (2+7)

([...], c) ([...], c)

If

```
if 2 then
  let ans=3+4 in
  print ans; ans
```

```
type i_expr =
  | IInt of int
  | IVar of string
```

```
type c_expr =
  | CInc of i_expr
  | CAdd of i_expr * i_expr
  | CIExpr of i_expr
  | ....
  | CIf of i_expr *a_expr *a_expr
```

```
type a_expr =
  | ALet of string * c_expr * a_expr
  | ACExpr of c_expr
```

```
if 2 then 3+4 else 5-1
```

```
([], 2)
([("x",3);("y",4)], x+y)
([("z",5);("w",1)], z-w)
```

```
let x=3 in
let y=4 in
let a= x+y in
let z=5 in
let w=1 in
let b=z-w in
if 2 then a else b
```

Compiling If

eax ≠ 0

```
here:
    cmp  eax, 0
    jne  here
```

cmp - compare

jmp - unconditional jump

jl - jump if <

jg - >

jle - ≤

jge - ≥

jne - ≠

je - =

jo - overflow

```
(if x then
    3
 else
    4)
+
(if x then
    3
 else
    4)
```

XLabel of string

```
    mov  eax, [esp-4]
    cmp  eax, 0
    je   else4
    mov  eax, 3
    jmp  after5
else4:
    mov  eax, 4
after5:
    mov [esp-8], eax
    mov  eax, [esp-4]
    cmp  eax, 0
    je   else6
    mov  eax, 3
    jmp  after7
else6:
    mov  eax, 4
after7:
    mov [esp-12] eax
    mov eax, [esp-8]
    add eax, [esp-12]
```