Considerando o esquema de tradução L-atribuído abaixo e sabendo que os operadores `*` e `&&` tem maior precedência:

S → if { B.t=rot(); B.f=S.next; } (B) { S_1.next=S.next; } S_1 { S.code=B.code || gera(B.t:) || S_1.code }

S → if { B.t=rot(); B.f=rot(); } (B) { S_1.next=S.next; } S_1 else { S_2.next=S.next; } S_2
{ S.code=B.code || gera(B.t:) || S_1.code || gera(B.f:); || S_2.code }

S → while { B.f=S.next; B.t=rot(); } (B) { S.beg=rot(); S_1.next=S.beg; }
S_1 { S.code=gera(S.beg:) || B.code || gera(B.t:) || S_1.code || gera(goto S.beg) }

S → **nome** = E; { S.code = E.code; || gera(nome.local = E.local); }

B → { B_1.t=B.t; B_1.f=rot(); } B_1 or { B_2.t=B.t; B_2.f=B.f; } B_2 { B.code=B_1.code || label(B_1.f) || B_2.code }

B → { B_1.t=rot(); B_1.f=B.f; } B_1 and { B_2.t=B.t; B_2.f=B.f; } B_2 { B.code=B_1.code || label(B_1.t) || B_2.code }

B → not { B_1.t=B.f; B_1.f=B.t; } B_1 { B.code=B_1.code; }

B → (B_1) { B.code=B_1.code; B.t=B_1.t; B.f=B_1.f; }

B → true { B.code=gera(goto B.t); }

B → false { B.code=gera(goto B.f); }

B → E_1 relop E_2 { B.code=E_1.code || E_2.code || gera(if E_1.local relop.lexval E_2.local goto B.t) || gera(goto B.f); }

E → $E_1 + E_2$ { E.local = temp(); E.code = E_1.code || E_2.code || gera(E.local = E_1.local + E_2.local); }

E → $E_1 * E_2$ { E.local = temp(); E.code = E_1.code || E_2.code || gera(E.local = E_1.local * E_2.local); }

E → (E_1) { E.local = E_1.local; E.code = E_1.code }

E → **id** { E.local = id.lexval; E.code = ""; }

Gere o código TAC para o trecho de código seguinte considerando o esquema de tradução acima:

1.
```
if (a < b && c < d)
    x = 2
```

2.
```
if (a < b) x = 2
while (c < d) y = 3
if (e < f) z = 4
```

3.
```
while (a < b + 3 && (e - 2) != f)
  if (c < d)
    x = y + z;
  else
    if (a > d)
      x = x - z;
```