De RAT à TAM

Exercice 1

Code RAT :

```
Prog {
    const a = 8;
    rat x = [6/a];
    int y = (a+1);
    x = (x + [3/2]);
    while (y < 12) {
        rat z = (x * [5/y]);
        print z;
        y = (y + 1);
    }
}
```

Code TAM correspondant :

```
prog:
    PUSH 2
    LOADL 6
    LOADL 8
    STORE (2) 0[SB]
    PUSH 1
    LOADL 8
    LOADL 1
    SUBR IAdd
    STORE (1) 2[SB]
    LOAD (2) 0[SB]
    LOADL 3
    LOADL 2
    CALL (-) RAdd
    STORE (2) 0[SB]
boucle:
    LOAD (1) 2[SB]
    LOADL 12
    SUBR ILss
    JUMPIF (0) finBoucle
    PUSH 2
    LOAD (2) 0[SB]
    LOADL 5
    LOAD (1) 2[SB]
    CALL (-) RMul
    STORE (2) 3[SB]
    CALL (-) ROut
    LOAD (1) 2[SB]
    LOADL 1
    SUBR IAdd
    STORE (1) 2[SB]
    JUMP  boucle
finBoucle:
    HALT
```

Passe de génération de code

Exercice 2

Actions à réaliser lors de la passe :

```
    let rec generation_code_expression e =
    match e with
    | AstType.AppelFonction(ia, listExp) ->
    | AstType.Ident(ia) ->
    | AstType.Booleen(bool) ->
    | AstType.Entier(int) ->
    | AstType.Unaire(un, expr) ->
    | AstType.Binaire(bin, expression1, expression2) ->

    let rec generation_code_instruction ia i =
    match i with
    | AstType.Declaration(ia, e) ->
    | AstType.Affectation(ia, e) ->
    | AstType.AffichageInt(e) ->
    | AstType.AffichageRat(e) ->
    | AstType.AffichageBool(e) ->
    | AstType.Conditionnelle(c,t,e) ->
    | AstType.TantQue(c,b) ->
    | AstType.Retour(e) ->
    | Empty -> ""

    and generation_code_bloc ia li =

    let generation_code_fonction (AstPlacement.Fonction(ia,_,li))  =
```