Placement Mémoire

Exercice 1 :

- f3 :
  - @a = -4[LB]
  - @b = -3[LB]
  - @r = -2[LB]
- f2 :
  - @b = -5[LB]
  - @x = -4[LB]
  - @y = -2[LB]
  - @x1 = 3[LB]
  - @x2 = 4[LB]
  - @res = 5[LB]
- f1 :
  - @i = -4[LB]
  - @r = -3[LB]
  - @n = -1[LB]
  - @r1 = 3[LB]
- test :
  - @x = 0[SB]
  - @y = 1[SB]
  - @z = 3[SB]

Exercice 2 :

```
module AstPlacement =
struct

(* Expressions existantes dans notre langage *)
(* = expression de AstType  *)
type expression = AstType.expression

(* instructions existantes dans notre langage *)
(* = instructions de AstType  *)
type bloc = AstType.bloc
 and instruction = AstType.instruction

(* informations associées à l'identificateur (dont son nom), liste de
paramètres, corps, expression de retour *)
(* Plus besoin de la liste des paramètres mais on la garde pour les tests
```

```
du placements mémoire *)
type fonction = Fonction of Tds.info_ast * Tds.info_ast list * bloc

(* Structure d'un programme dans notre langage *)
type programme = Programme of fonction list * bloc
```

Exercice 3 :

```
let rec analyse_placement_instruction reg dep i =
  match i with
  | AstType.Declaration (ia, _) ->
  | AstType.Conditionnelle (_,t,e) ->
  | AstType.TantQue (_,b) ->
  | _ -> (i, dep)

and analyse_placement_bloc reg dep li =
  match li with
  | [] -> []
  | i::q ->
    let (ni, ndep) = analyse_placement_instruction reg dep i in
      ni::(analyse_placement_bloc reg ndep q)

let rec analyse_placement_param dep rlp =

let analyse_placement_fonction (AstType.Fonction(ia,lp,li))  =

let analyser (AstType.Programme (fonctions,prog)) =
```