TD6
TdL

Traduction des langages

Les pointeurs

Objectif :
- Etre capable d'ajouter les pointeurs au langage RAT
- Modifier le compilateur en conséquence

1 Les pointeurs

Une variable est physiquement identifiée de façon unique par son **adresse**, c'est-à-dire l'adresse de l'emplacement mémoire qui contient sa valeur.
Un **pointeur** est une variable qui contient l'**adresse** d'un autre objet informatique (une "variable de variable" en somme).

1.1 Déclaration

La déclaration d'un pointeur se fait selon la syntaxe suivante :

```
type* id;
```

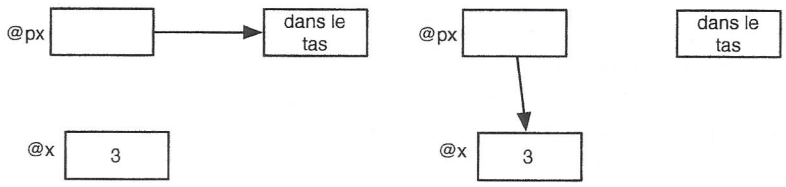
Cette instruction déclare une variable de nom id et de type pointeur(type) (pointeur sur une valeur de type type). Par exemple :

```
int* x;
```

déclare une variable x qui pointe sur une valeur de type int.

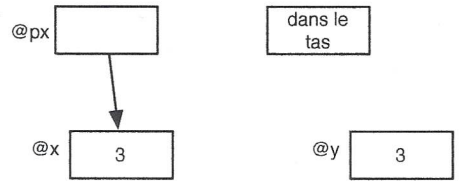
1.2 Adresse et valeur pointée

Les deux opérateurs particuliers en relation avec les pointeurs sont : & et *.
- & est l'opérateur qui **retourne l'adresse mémoire d'une variable**
  Si x est de type t alors &x est de type Pointeur(t)
  Exemple :
  ```
  int * px = (new int);
  int x = 3;
  px = &x;
  ```



- * est l'opérateur qui **retourne la valeur pointée par une variable pointeur.**
  Si x est de type Pointeur (t) alors *x est de type t
  Exemple :

```
int * px = (new int);
int x = 3;
px = &x;
int y = *px;
```



1.3 Exercice

▷ **Exercice 1** *Donner le type des variables dans le programme suivant :*

```
int * x = (new int);
int * * y = & x;
int z = 3;
*x = z;
*y = &z;
```

▷ **Exercice 2** *Donner le type des variables dans le programme suivant :*

```
pointeur{
  int * x = (new int);
  * x = 3;
  int z= 18;
  int *y = & z;
  * y = *x;
}
```

2 Modification de TAM et du compilateur

▷ **Exercice 3**

1. *Quelle(s) règles(s) faut-il modifier ou ajouter pour compléter l'introduction des pointeurs dans le langage ?*
2. *Quelles modifications faut il apporter à l'AST ?*
3. *Modifier la passe de gestion des identifiants.*
4. *Modifier la passe de typage.*
5. *Modifier la passe de placement mémoire.*
6. *Proposer la traduction en TAM de l'exemple de l'exercice 2. On rappelle l'existence des instructions TAM suivantes, permettant de manipuler des adresses :*
   - *Empiler une adresse : LOADA d[r]*
   - *Empiler n mots à partir de l'adresse laissée en sommet de pile : LOADI (n)*
   - *Écrire n mots de la pile à l'adresse laissée en sommet de pile : STOREI (n)*
   - *Allocation de mémoire : SUBR Malloc (réserve dans le tas une zone de la taille laissée en sommet de pile, l'adresse obtenue est laissée en sommet de pile).*
7. *Modifier la passe de génération de code.*

TD6
TdL

Les pointeurs

Exercice 1 : Type

x : pointeur sur un entier
y : pointeur sur un pointeur sur un entier
z : entier

Exercice 2 : Type

x : pointeur sur un entier
z : entier
y : pointeur sur un entier

Exercice 3 : Modification de TAM et du compilateur

1/ I → A = E (A = affectable)
A → id
A → (*A)
TYPE → TYPE*
E → A
E → null
E → (new TYPE)
E → &id

IP faut ajouter le * et le & à l'analyseur lexical et syntaxique.

Ajouter les règles dans menhir.

2/ AST Syntax / ASTTds / ASTType / ASTDep / Génération de code
3/
4/ type instruction =
5/ | Declaration of typ x ~~string~~ info_ast x expression
7/ | Affectation of ~~string~~ affectable x expression → code exp STORE (taille typ) @ → code exp code affectable STOREI
: :

type expression =
| ~~Ident of string~~
| Numerateur of int
| Affectable of affectable
| Null
| New of typ info_ast
| Adresse of ~~string~~
: :

SUBR Mabib <=> LOADL 0
LOADL (taille typ)
SUBR Malloc

↳ LOADA (@info_ast)

type affectable = info_ast charge l'@ → de l'affectable en sommet de pile
| Ident of ~~string~~
| Valeur of affectable

+ analyser_tds_affectable
+ rien à faire...
+ rien à faire, juste définir la taille d'un pointeur à 1.

type typ = Int | Bool | Rat | Undefined | Pointeur of typ

4/ Jugement de typage

- $\dfrac{\sigma \vdash a : \text{Pointeur}(\tau)}{\sigma \vdash *a : \tau}$
- $\sigma \vdash \text{null} : \text{Pointeur}(\text{undefined})$
- $\dfrac{\sigma \vdash t : \tau}{\sigma \vdash \text{new } t : \text{Pointeur}(\tau)}$
- $\dfrac{\sigma \vdash x : \tau}{\sigma \vdash \&x : \text{Pointeur}(\tau)}$
- $\dfrac{\sigma \vdash t : \tau}{\sigma \vdash t* : \text{Pointeur}(\tau)}$

```
(* ASTTds.Affectable -> ASTType.Affectable x typ *)
let rec analyser_type_affectable a =
  match a with
  | Valeur p ->
      let (np, tp) = analyser_type_affectable p in
      (match tp with
       | Pointeur(t) -> (Valeur np, t)
       | _ -> raise ... )
  |...
```

+ compléter la fonction est_compatible sur les types.

5/ Placement mémoire

6/ Génération de code

```
int *x = (new int);
*x = 3;
int z = 18;
int *y = &z;
*y = *x;
```

```
PUSH (-1)
LOADL 1
SUBR MAlloc
STORE (-1) 0[SB]
LOADL 3
LOADA 0[SB]
STOREI (-1)
PUSH 1
LOADL 18
STORE (-1) 1[SB]
PUSH 1
LOADA 1[SB]
STORE (-1) 2[SB]
LOAD (-1) 0[SB]   //x
LOADI (-1)        //*x
LOAD (-1) 2[SB]   //y
STOREI (-1)
```

7/

```
int y = 3
y = 4
```

```
PUSH 1
LOADL 3
STORE (-1) 0[SB]
LOADL 4
STORE (-1) 0[SB]
LOADA 0[SB]
STOREI (-1)
```