Rapport projet de TDL

Maxence Ahlouche Martin Carton
Clément Hubin-Andrieu

mai 2014

Table des matières

1 Design decisions **2**
1.1 Typage 2
1.2 Noms de types 2

2 NULL et nil, YES et NO **2**

3 Classes **2**

4 Extension du langage **2**
4.1 Alias de type 2
4.2 Tableaux 2
4.3 Opérateurs new et delete 3
4.4 Opérateur sizeof 3

5 Warnings **3**
5.1 unreachable 3
5.2 shadow 3

1 Design decisions

1.1 Typage

Le typage est plus fort qu'en C. Ça empêche notamment d'écrire une fonction comme `malloc`, mais on a un opérateur `new` (voir section 4.3) qui fait ça très bien.

1.2 Noms de types

Les noms de type commencent tous par une majuscule. Ceci afin de permettre d'avoir des alias de type (voir section 4.1).

2 NULL et nil, YES et NO

Il n'y a pas de `nil` (qui serait inutile vu qu'il n'y aurait pas de différence avec `NULL`) et `NULL` s'écrit `null` par consistance avec les autres variables.

De même `YES` et `NO` s'écrivent `yes` et `no`.

3 Classes

Nous avons supprimé les `@` devant `@class` et `@end`.

4 Extension du langage

4.1 Alias de type

Il est possible de définir des alias de type :

```
using NouveauNom = NomExistant;
```

La syntaxe évite volontairement le `typedef` bizarre du C.

4.2 Tableaux

On peut créer des tableaux :

```
Char[5] s = "net7";
```

La taille se met après le type, et non le nom comme en C.

Ils sont convertibles en pointeurs vers le type correspondant, mais le font dans moins de cas qu'en C (notamment les tableaux sont copiables et passables comme paramètre de fonction et peuvent être retournés). Le type tableau est un vrai type.

4.3 Opérateurs new et delete

Ces opérateurs sont équivalents à `malloc` et `free`, mais sont typés (le langage ne possède pas de type `void*`).

```
Char* test = new(Char);
delete(test);
```

4.4 Opérateur sizeof

Cet opérateur retourne la taille d'un type :

```
Int a = sizeof(Int);
```

5 Warnings

Nous avons ajoutés des warnings au compilateur.
Il suffit d'appeler *mocc* avec `-w nom_du_warning`. Il y a aussi un warning `all`.

5.1 unreachable

`unreachable` vérifie la présence d'instructions inutiles.
Par exemple :

```
Int test() {
    if(a) {
        return 123;
    }
    else {
        return 456;
    }
    f(); // unreachable
}
```

5.2 shadow

`shadow` vérifie qu'une déclaration ne masque pas une déclaration précédente.