Enseirb-Matmeca
Filière Informatique

Projet de compilation
Spécification du compilateur

Auteurs :
Florian Le Vern
Sylvain Cassiau

17 novembre 2015

1 Définition du langage accepté par le compilateur

Le langage que notre compilateur devra reconnaître s'apparente fortement à du `C`.
Il n'utilise pas de pointeurs explicitement (c'est à dire qu'on ne verra pas apparaître le caractère `*` du `C`), ni de structures et d'union.
Il gère cependant les tableaux à plusieurs dimensions suivant la notation `A[i][j][k]`. Le tableaux sont des références "cachées" et peuvent être affectés par référence comme suit : `int A[10]; int B[] = A;`
Les variables ont une portée de bloc délimité par des accolades comme en `C`, et elles ne peuvent pas prendre des noms réservés comme `for`, `int` ou `map` par exemple.

1.1 Les types

Le langage reconnu par notre compilateur prendra en compte les types suivants : `int, float, char` qui sont des types signés. Les booléens seront associés aux entiers (0 faux ; $n \neq 0$ vrai).
Le langage permettra de stocker/affecter et passer en paramètre des références de fonctions dont le type est noté `typeRetour (typeArg1, typeArg2, ...)`.
Ainsi que des tableaux d'objets de types précédents.

Voici un tableau récapitulant les types de retour à la suite d'opérations entre différents types. Les opérations non précisées sont *a priori* interdites.

+, -, *, /	char	int	float
char	char	int	float
int	int	int	float
float	float	float	float

On notera que les casts ne sont pas autorisés pour les tableaux ou les fonctions. Les comparaisons (égalité, inégalités) sont permises entre les char, int, et float et donnent les résultats bien connus des grands mathématiciens quant à leur ordre ($3 < 3.1415926535$ par exemple).

1.2 Les structures d'exécution

Le langage que reconnaît notre compilateur comporte les structures d'exécution `for`, `while`, `do-while`, `if-else if-else` et `switch`.

1.3 Les fonctions externes

Le langage dispose de quelques fonctions qui lui sont propres et dont certaines sont optimisées par le compilateur :

`typeA[] map(f, typeB[]) :` Applique une fonction unaire `typeA f(typeB)` à tous les éléments d'un tableau à une dimension, et retourne le tableau de même taille contenant les résultats. Cette opération est vectorisée par le compilateur pour optimiser les performances.

`typeA reduce(f, typeA[]) :` Applique une fonction d'arité 2 **associative** sur l'ensemble des éléments d'un tableau à une dimension pour le réduire. Le calcul est là aussi vectorisée.

`void print(char[]) :` Affiche sur la sortie standard un tableau de caractères (en l'associant à la chaîne de caractères correspondante).

`void printchar(char) :` Affiche sur la sortie standard un caractère.

`void printint(int) :` Affiche sur la sortie standard un entier.

`void printfloat(float) :` Affiche sur la sortie standard un flottant.

`void free(typeA[]) :` Libère de la mémoire allouée. Le tableau devient une référence ne pointant sur aucun espace mémoire. Il peut donc être re-affecté ensuite.

`int size(typeA[]) :` Retourne le nombre d'éléments du tableau. (Et non le nombre d'octets utilisé par son contenu).

Le langage supporte par ailleurs les opérateurs suivants :
— `+, -, *, /` : somme, différence, multiplication et division des types classiques
— `%` : modulo entre entiers
— `||, &&, !` : ou, et, non. Opérations logiques entre entiers en tant que booléens.
— `<, <=, >, >=, ==, !=` : Comparaisons entre les types arithmétiques. Les tests d'égalité et de différence existent aussi pour comparer les références de fonctions et de tableaux.
— `=, *=, /=, %=, +=, -=` : Opérateurs d'affectation

1.4 Les tableaux

Les tableaux du langage sont des entités particulières. En effet, ils peuvent être créés sans taille (`int A[];`) et correspondent alors à une référence ne pointant vers aucun tableau en mémoire (une sorte de pointeur NULL). Ils peuvent être affectés entre eux (c'est d'ailleurs la seule opération permise sur les tableaux, à part l'accès `A[i]`). On copie alors la référence d'un tableau dans l'autre, suite à quoi ils pointent tous deux vers le même espace mémoire. La seule façon de créer des tableaux en mémoire est de le faire dynamiquement en déclarant leur taille (`int B[10];`). La référence pointe alors vers un nouvel espace mémoire de la taille précisée. Dans tous les cas, la taille des espaces mémoire est constante. Ainsi, dans le code ci-dessous, les tableaux `A` et `B` ont à la fin une taille de 100 et référencent le même tableau :

```
int A[];
int B[100];
A = B;
```

Nous essayerons dans la mesure du possible d'implémenter un ramasse-miettes pour désallouer les tableaux qui ne sont plus référencés. Si cela s'avère trop complexe, nous fournirons une fonction `free(typeA A[])` pour libérer la mémoire.

2 Descriptions des tests

Les tests présents dans le répertoire `/tst/tst_unit/` serviront au cours du développement comme *pseudo* tests unitaires, afin de vérifier le fonctionnement des fonctionnalité du langage au fur et à mesure de leurs implémentations. Chaque test se veut minimaliste et démontre une ou quelques unes des fonctionnalité du compilateur. Ainsi, lors du développement de ce dernier, nous pourrons nous baser sur ces tests afin de jauger notre avancement sur le projet.
Les tests présents dans le répertoire `/tst/tst_recette/` sont des tests plus conséquents qui vérifient plusieurs aspects du compilateur. Il s'agit de plus gros programmes visant à démontrer le fonctionnement du compilateur dans l'ensemble de ses fonctionnalités. Ils devront eux aussi fonctionner comme attendu à la fin du projet.

Les tests se trouvent dans les dossiers `/tst/*`. Chaque fichier commence par la description de ce que teste le test, le résultat attendu à la compilation, et le résultat attendu à l'exécution.