Premier rapport - 13/11/2015
Compilateur C Map/Reduce pour multicoeurs

Groupe 3 - PG203
Programmeurs : ohayak, desparbes

Equipe

L'équipe est constituée de HAYAK Omar, et d'ESPARBES Romain, du groupe 3 de travaux dirigés, avec M. SERGENT Marc comme encadrant.

1 Implémentation

Notre implémentation est basée sur le langage C. les fonctionnalités supportées sont présentées ci-dessous. Elles sont globalement ressemblantes à celles indiquées sur le site du projet, et nous ne souhaitons pas ajouter de fonctionnalités supplémentaires.

1.1 Types supportés

Les types supportés sont :

- *float*
- *int*
- les pointeurs de fonctions
- tableaux statiques et dynamiques de *float*
- tableaux statiques et dynamiques de *int*
- Le compilateur gère les variables et tableaux globaux.

1.2 map, reduce et autres fonctions

- On peut déclarer les fonctions externes grâce à des prototypes.
- Les paramètres peuvent être des *int*, des *float* et des tableaux dynamiques.
- Les fonctions *printint(int)* et *printfloat(float)* sont prédéfinies et peuvent être employées par l'utilisateur.
- Les fonctions *reduce()* et *map()* sont reconnues par la grammaire.
- La fonction `map` a un type générique, dont le prototype est :
  `TYPE2 (map(TYPE2 fct(TYPE1),TYPE1 array[N]))[N]`
  avec `TYPE1` et `TYPE2` n'importe quel type valide, sauf `VOID`. La fonction `map` prend en paramètre un tableau `array` de `N` éléments de type `TYPE1` et une fonction `fct` qui transforme un élément de type `TYPE1` en `TYPE2`. Le type de retour de `map` est un tableau d'éléments de `TYPE2`. La fonction `map` doit appliquer la fonction `fct` en parallèle à tous les éléments du tableau d'entrée.
- La fonction `reduce` est également générique et a pour prototype :
  `TYPE reduce(TYPE fct(TYPE a, TYPE b), TYPE array[])`
  avec `TYPE` n'importe quel type valide, sauf `VOID`. Elle applique la fonction `fct` sur tous les éléments du tableau `array`. C'est une réduction, comme lorsque l'on fait une somme entre tous les éléments d'un tableau. La fonction `fct` est supposée être associative : le parenthésage n'a pas d'importance pour le calcul du résultat.

1.3 Opérations

Le compilateur gère les opérations courantes telles que "+", "-" , "*", " !","<",">","<=",">=","!=" et "==" entre deux variables de même type, sinon, on applique les règles de conversion de type standard.

1.4 Instructions

Le compilateur gère :

- les affectations
- les casts automatiques *int* vers *float* et *float* vers *int*
- affectations entre tableaux
- les boucles *for*, *while* et *do...while*
- le *if* (qui peut avoir ou non un *else*).
- les blocs d'instructions.

1.5 Récursivité

Ce compilateur gèrera la récursivité nativement.

2 Codage du compilateur

Le compilateur sera implémenté en C, pour être au plus proche du code et pour ne pas s'emcombrer avec la notion d'objet, qui ne nous pas primordiale dans un projet de compilation.

2.1 Gestion des erreurs

La justesse du code doit être vérifiée syntaxiquement et sémantiquement :

- les types interdits,
- variables ou fonctions non déclarées,
- affectation d'une variable d'un mauvais type,
- un tableau ne peut prendre que la valeur d'un tableau plus grand,
- des types non correspondants pour les opérateurs,
- non concordance des arguments lors de l'appel d'une fonction,
- fonction void qui retourne une valeur,
- fonction qui retourne un type différent du type de retour déclaré.

3 Tests

Nous avons 24 tests au total au moment d'écrire ce rapport. Malheureusement, ils balaient principalement la compilation de programmes corrects. Nous comptons ajouter des tests sur la gestion des erreurs au fur et à mesure de l'avancement de notre projet.

4 Conclusion

Pour ce compilateur, nous nous sommes limités à l'essentiel, pour être sûr d'avoir un outil fonctionnel et testé convenablement à la fin de l'échéance. Il est possible que nous rajoutions des fonctionnalités, mais nous fairons évidemment en sorte que si le langage évolue, ce premier jeu de test reste valide.