CMP1 – Construction des compilateurs
TYLA – Typologie des Langages

EPITA – Promo 2011 (ERASMUS)
Tous documents (notes de cours, polycopiés, livres) autorisés
Calculatrices et ordinateurs interdits.

(1h30)

Une copie synthétique, bien orthographiée, avec un affichage clair des résultats, sera toujours mieux notée qu'une autre demandant une quelconque forme d'effort de la part du correcteur. Une argumentation informelle mais convaincante, sera souvent suffisante. Une lecture préalable du sujet est recommandée.

1 Énumérations fortement typées

Dans cet exercice, on s'intéresse à l'ajout d'énumérations (mot clef `enum` en C++) fortement typées dans le langage Tiger. Nous allons notamment passer en revue les différentes parties de notre compilateur afin de déterminer les aménagements nécessaires.

Rappel : le langage Tiger utilisé dans cet exercice est celui qui est décrit dans le Manuel de Référence du Compilateur Tiger, qui est utilisé comme référence tout au long du projet du même nom.

1.1 Préliminaires

1. À quoi peuvent servir les énumérations ? Donnez un ou deux exemples.
2. Quel est le défaut des énumérations en C++ ?
3. Avant d'étendre le compilateur, réfléchissons à la forme que nous souhaiterions donner à cette extension. Donnez un exemple court mais complet de ce que vous souhaiteriez pouvoir écrire avec une telle extension pour les énumérations. Votre exemple devrait au moins comporter
   – une déclaration d'un type énumération ;
   – une déclaration d'une variable de ce type ;
   – une utilisation de cette variable.

1.2 Partie frontale (*front end*)

Cette partie s'intéresse aux modifications à apporter au front end Tiger pour supporter les énumérations.

1. **Spécification lexicales.** Que faut-il ajouter aux *spécifications* lexicales du langage pour supporter les énumérations ?
2. **Scanner.** Comment étendre le scanner, c'est-à-dire, que faut-il ajouter/modifier dans le fichier '`scantiger.ll`' pour supporter ces nouvelles spécifications lexicales ?
3. **Spécification syntaxiques.** Que faut-il ajouter aux *spécifications* syntaxiques du langage pour supporter les types énumérations ?

4. **Syntaxe abstraite.** Faut-il effectuer des modifications ou des ajouts dans la syntaxe abstraite du langage pour supporter les énumérations ? Si oui, lesquelles ?
5. **Parser.** Comment étendre le parser, c'est-à-dire, que faut-il ajouter/modifier dans le fichier '`parsetiger.yy`' pour supporter ces nouvelles spécifications syntaxiques ?
6. **Liaison des noms.** Que faut-il changer dans le `Binder` vis-à-vis des énumérations ?
7. **Vérification des types.** Qu'allez-vous changer dans le module `type` du compilateur ?
8. **Représentation intermédiaire** Faut-il ajouter/modifier quelque chose dans le langage Tree ?
9. **Traduction (vers IR).** Y'a-t-il des changements à apporter au visiteur chargé de la traduction vers la représentation intermédiaire ? Justifiez votre réponse.

1.3 Partie terminale (*back end*)

Nous n'avons pas étudié le back end de `tc` à ce stade de l'année, mais nous avons déjà un évoqué ses composants, et vous avez déjà été sensibilisés à des notions connexes dans d'autres cours (assembleur, architecture des ordinateurs, etc.).

Grossièrement, le back end de `tc` effectue deux tâches :

L'allocation de registres Chaque fonction peut contenir un nombre arbitraire de variables ou *temporaires* ; or un processeur n'a qu'un nombre fini de *registres*. L'allocateur de registres a pour tâche de résoudre ce problème en faisant appel à la mémoire au besoin.

La génération de code en langage d'assemblage Il s'agit de la dernière étape de traduction, qui fait le pont entre la représentation intermédiaire (en Tree) et la sortie (en langage d'assemblage MIPS ou IA-32 dans notre cas).

1. **Allocation de registres.** Est-ce que l'adjonction d'énumérations à Tiger modifie l'allocateur de registres ?
2. **Génération de code.** De même, la génération de code à partir de la représentation intermédiaire est-elle affectée ?

1.4 La question bonus

Quelle extension concomitante de celles des énumérations pourrait-on souhaiter dans Tiger ?

2 Booléens

Dans cette partie, on étudie une application des types énumérés implémentés dans la section précédente, concernant les types booléens.

1. Les booléens peuvent être vus comme des types énumérés. Quel peut-être l'intérêt de les fournir dans la bibliothèque standard du langage sous forme d'énumération (plutôt que de les confondre avec les entiers, comme c'est le cas actuellement) ? Attention, il ne s'agit pas d'une extension supplémentaire (en sus de celle des énumérations).
2. Qu'ajouteriez vous (types, fonctions, etc.) au prélude de `tc` pour étendre la bibliothèque standard avec un type énuméré représentant les booléens ? Vous pouvez soit fournir du code annoté, soit décrire votre solution sous forme de description.
3. Quel est le défaut de cette approche ?