UFR 919 Informatique – Master Informatique

Spécialité STL – UE MI016 – ILP

TME3 — Extension de la bibliothèque d'exécution

Christian Queinnec

Objectif : Étendre la bibliothèque d'exécution d'ILP1 avec des fonctions et un nouveau type de données : le vecteur.

Buts

— Ajouter une primitive à l'interprète
— Ajouter une primitive au compilateur
— Ajouter une primitive à la bibliothèque d'exécution (en C)
— Ajouter un nouveau type de données à la bibliothèque d'exécution (en C)
— Écrire un nouveau patron C

Vous aurez à suivre le mémento en `http://www-master.ufr-info-p6.jussieu.fr/2013/Ext/queinnec/ILP/refcard.pdf`. Vos fichiers Java seront donc dans le paquetage `fr.upmc.ilp.tme3`. Vos fichiers C seront dans le répertoire C.

1 Une nouvelle primitive : sinus

En ILP1, il n'est pas possible de calculer directement le sinus d'un nombre. On souhaite dans cet exercice ajouter une primitive, qui sera nommée `sinus` en ILP, pour pouvoir effectuer ce calcul. On essaiera d'adopter une approche générale qui permettra d'ajouter simplement par la suite d'autres fonctions si on le souhaite.

1. Quelles sont les grandes étapes des modifications à apporter ?
2. Implanter ces modifications.

Remarque : vous pouvez vous inspirer de l'implantation des primitives `print` ou `newline` qui sont présentes dans ILP1.

On fera attention à la comparaison des valeurs de retour flottantes dans l'interprétation et la compilation. En effet, le formatage en Java et en C peut être différent [1]

Comme pour les précédents travaux, les extensions ne doivent pas modifier le code existant mais l'étendre.

1. On pourra utiliser la classe `DecimalFormat` en Java pour obtenir des flottants dans le même format qu'en C.

2 Un nouveau type : les vecteurs

On souhaite maintenant pouvoir gérer des données dans des vecteurs. Il faut donc ajouter ce nouveau type dans ILP. À ce type, on ajoutera trois nouvelles primitives qui auront pour noms en ILP : `make-vector`, `vector-length`, `vector-get` (attention aux tirets qui ne sont pas des blancs soulignés) dont voici les signatures plus précises :

```
1 make-vector(taille, objet)
2 vector-length(vecteur)
3 vector-get(vecteur, index)
```

La primitive `make-vector` crée un vecteur ayant pour taille son premier argument, chaque cellule de ce vecteur sera initialisée avec le second argument.

La primitive `vector-length` renvoie la taille du vecteur qu'elle reçoit en argument.

La primitive `vector-get` renvoie le index-ième objet du vecteur.

Comme ILP1 est sans effet de bord, on n'ajoutera pas `vector-set` qui permettrait d'écrire dans le vecteur.

1. Quelles sont les grandes étapes des modifications à apporter ? Suivre les indications de la question précédente.
2. Implanter ces modifications.

   Indication : Vous regarderez les directives de compilation conditionnelles d'`ilp.h` pour imposer votre nouvelle définition de `ILP_Object`.