IFT580 - Compilateur

Implémentation du devoir pour IFT580 - Compilation et interprétation des langages

Premier pas

Ces instructions vous fourniront une copie du projet opérationnel sur votre ordinateur local à des fins de développement et de test.

Prérequis

- `java` doit être installé et accessible via les variables d'environnement (vous devez pouvoir utiliser la commande `java` dans un terminal).
- `.NET Core 3.0` doit être installé. Sous Windows, Visual Studio 2019 devrait s'en charger. Sur Ubuntu 20.04, utilisez les commandes suivantes:

  ```
  wget -q https://packages.microsoft.com/config/ubuntu/20.04/packages-
    microsoft-prod.deb -O packages-microsoft-prod.deb
  sudo dpkg -i packages-microsoft-prod.deb
  sudo add-apt-repository universe
  sudo apt-get update
  sudo apt-get install apt-transport-https
  sudo apt-get update
  sudo apt-get install dotnet-sdk-3.0
  ```

  Assurez-vous que la commande `dotnet` fonctionne. D'autres versions de `.NET Core` peuvent fonctionner, mais n'ont pas été testées.
- `Clang` doit être installé ainsi que la runtime `LLVM`. Ils devraient tous être accessible via le path. Sur Windows, Cygwin est la façon la plus simple de les installer. Sur Ubuntu:

  ```
  sudo apt install llvm-runtime clang
  ```

Compilation

Générez simplement la solution dans Visual Studio 2019 ou utilisez la commande suivante :

```
dotnet build Ccash.sln
```

Par défaut, l'algorithme SNCAG est exécuté à chaque fois que le projet est compilé. Puisque cela prend relativement beaucoup de temps, il pourrait être souhaitable de désactiver l'exécution automatique en changeant la valeur de `SNCAGEnabled` à `false` dans `Ccash/Ccash.csproj`.

L'algorithme SNCAG peut être exécuté manuellement en tant que test unitaire dans le projet `Ccash.SemanticAnalysis.tests`.

Compilation d'un des fichiers tests en C$

```
cd Ccash
dotnet run test_files/main.ccash
```

Alternativement, je recommande d'utiliser le bouton lancer dans Visual Studio ou Rider. Il est beaucoup plus facile de debuger des erreurs et des exceptions ainsi. Le fichier qui se fait compiler peut être changé à même l'IDE, ou bien le fichier `Ccash/Properties/launchSettings.json` peut être modifié.

Il est important de noter que les tests contenus dans ces fichiers sont **non exhaustifs**. Ce ne sont que des fichiers temporaires qui ont été utilisés pour nous convaincre que l'implémentation de différentes fonctionnalités était correcte pour certains cas-limites.

Nous pensions que ces fichiers pourraient vous être d'une certaine utilité, mais vous ne devriez pas vous fier entièrement dessus.

Commandes utiles

Exécuter le bytecode `LLVM` produit par le compilateur:

```
lli main.bc
```

Cela pourrait ne pas fonctionner sur Windows. Dans ce cas, simplement faire:

```
clang main.bc
a.exe
```

Visualisation du `LLVM IR` produit par le compilateur dans un terminal:

```
llvm-dis -o - main.bc | less
```

ou

```
cat main.ll | less
```

ou simplement ouvrir `main.ll` dans votre éditeur de code favoris.

Support

Si vous avez besoin d'aide, vous pouvez me contacter n'importe quand à karim.elmougi@gmail.com ou k.elmougi@usberbrooke.ca