IFT580 - Compilateur

Assignment for IFT580 - Compilation et interprétation des langages

Getting Started

These instructions will get you a copy of the project up and running on your local machine for development and testing purposes.

Prerequisites

- `java` must be installed and present in path (we must be able to use the `java` command in a terminal).
- `.NET Core 3.0` must be installed. On Windows, Visual Studio 2019 should take care of it. On Ubuntu 20.04, use the following commands:

  ```
  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
  ```

  Make sure that the `dotnet` command works. Other versions of `.NET Core` may work, but haven't been tested.

- `Clang` must be installed, as well as the `LLVM` runtime. They should all be present in path. Only version 9 and up have been tested. On Windows, installing Cygwin is the easiest way to get them. On Ubuntu:

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

Building

Simply build the solution in Visual Studio 2019, or use the following command :

```
dotnet build Ccash.sln
```

By default, the SNCAG algorithm is executed each time the compiler is built. As this takes a fairly long time, you may wish to disable that behavior by setting `SNCAGEnabled` to `false` in `Ccash/Ccash.csproj`.

The SNCAG algorithm may be run manually at any time as a unit test in the `Ccash.SemanticAnalysis.Tests` project.

Compiling one of the C$ test files

```
cd Ccash
```

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

Alternatively, I would recommend just pressing Run or Debug in Visual Studio or Rider. It is much easier to debug errors and exceptions this way. You can change which file gets run from the IDE, or you can edit the `Ccash/Properties/launchSettings.json` file.

It is important to note that the tests contained in these files are **not exhaustive**. They are just scratch files that were used to reasonably convince us that the implementations of different functionalities were correct in certain edge cases.

We thought you may find them of some use, but you shouldn't rely entirely on them.

Useful commands

Executing the `LLVM` bytecode produced by the compiler:

```
lli main.bc
```

This may fail on Windows, simply use `clang` instead:

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

Viewing the `LLVM IR` produced by the compiler can be done in the terminal with:

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

or

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

or simply open `main.ll` in your favorite text editor.

Support

If you need help you can contact me at any time at karim.elmougi@gmail.com or k.elmougi@usberbrooke.ca