编译原理研讨课实验PR001-CACT说明

作业提交

每组同学Gitlab账号下有一个名为compiler的project，同学们需要把代码提交到自己账号下的该project下的master分支，检查时我们会从该project下clone代码，编译运行并打分。如：

```
git clone http://124.16.71.65/compiler1/compiler.git
```

其中的`compiler1`为组名，各小组根据自己的组名进行修改。

测试与打分

1. 最终的成绩由三部分组成：实验报告、测试样例通过率（功能分）、生成的汇编代码的性能（性能分）。
2. 测试用例分为两种
    1. 功能测试用例：大部分公开用例 + 一些隐藏用例（PR001-PR003均有）。
    2. 性能测试用例（仅出现在PR003）。
3. 公开测试用例与性能测试用例均会放在课程网站和Gitlab上。
4. 实验报告每组由固定的一个同学提交即可。
5. 词法语法分析，语义分析阶段会有本身就不能通过的用例，同学们编写的编译器要能够正确处理，中间代码生成和汇编代码生成都是可以正确编译出来的

熟悉ANTLR的安装和使用

第一步：登录到本组服务器的帐号，参见实验环境说明。

每个组有两个账户（其中`1`为组号，各小组根据自己的组名进行修改）：

1. 服务器登录：`compiler1:compiler1!`；
2. Gitlab账号：`compiler1:compiler1!compiler1!`；

第二步：验证安装JDK：

查看是否安装JDK及版本（版本应在1.6及以上）：

```
java -version
```

第三步：将源代码从Gitlab服务器clone到本地：

配置自己的git偏好设置:

```
git config --global user.name "compiler1"
git config --global user.email "compiler1xxx@xxxx.com"
git config --global core.editor vim
# vim or emacs, up to you
```

第四步：安装ANTLR

1. 下载ANTLR的Java Lib到本地

```
mkdir ~/lib
# git clone时输入的为上述gitlab的账号和密码
git clone http://124.16.71.65/compiler0/material.git
cp material/antlr-4.8-complete.jar ~/lib
```

2. 把`antlr-4.8-complete.jar`加入到`CLASSPATH`中。注意其中的路径名应为各自组对应的账户：

```
# 该命令仅在当前终端有效，想永久对该用户有效，需编辑~/.bashrc
$ export CLASSPATH=".:/home/compiler1/lib/antlr-4.8-complete.jar:$CLASSPATH"
# 使上述修改永久有效，编辑~/.bashrc，
$ vim ~/.bashrc
# 将上述命令添加到文件最后
$ source ~/.bashrc
```

3. 创建别名，编辑`.bashrc`，`source ~/.bashrc`使环境变量生效，注意其中的路径名应为各自组对应的账户：

```
$ alias antlr4='java -Xmx500M -cp "/home/comiler1/lib/antlr-4.8-
complete.jar:$CLASSPATH" org.antlr.v4.Tool'
$ alias grun='java -Xmx500M -cp "/home/compiler1/lib/antlr-4.8-
complete.jar:$CLASSPATH" org.antlr.v4.gui.TestRig'

# 使上述修改永久有效，编辑~/.bashrc，
$ vim ~/.bashrc
# 将上述命令添加到文件最后
$ source ~/.bashrc
```

检查安装结果，执行：

```
$ antlr4
```

看到如下输出时，说明ANTLR安装成功：

```
ANTLR Parser Generator  Version 4.8
......
......
```

至此，ANTLR设置完成。

第五步：运行antlr compiler demo

1. 拷贝代码（其中`compiler1`为组名，请同学们自行修改）：

```
$ cd ~
# git clone时输入的为上述gitlab的账号和密码
$ git clone http://124.16.71.65/compiler1/compiler.git
```

2. 编写语法文件：

```
$ cd compiler
$ vim grammar/CACT.g4
# 编写CACT.g4
# ......
# ......
```

3. 在`grammar`目录下执行

```
# 进入grammar目录
# cd grammar
$ antlr4 -Dlanguage=Cpp CACT.g4
```

生成parser的代码，默认放在当前目录下；

4. 编写`compiler/src/`下的文件，使用parser提供的接口对语法树进行操作。

编写完代码之后，在`compiler/`下编译：

```
$ mkdir build && cd build
$ cmake ..
$ make
```

运行测试样例：

```
$ ./compiler ../samples/00_main.cact
```