# 方案拟订任务
# 定于 2020 年 5 月 5 日星期二下午 11 时 59 分到期

## 1 导言

在这个作业中，您将为 Cool 编写一个解析器。 赋值使用了两个工具：一个称为 bison 的解析器生成器和一个用于操作树的包。 解析器的输出将是一个抽象语法树(AST)。 您将使用解析器生成器的语义操作构造此 AST。

您当然需要参考 Cool 的句法结构，如 Cool 参考手册（以及其他部分）的图 1 所示。 野牛的文档可以在网上找到，树包在“酷支持代码之旅”讲义中描述。 您将需要用于此任务和未来任务的树包信息。

这个讲义中有很多信息，你需要知道大部分信息才能编写一个工作解析器。 请仔细阅读讲义。

和以前一样，您可以单独工作，也可以作为一个对进行这项任务。

## 2 文件和目录

要开始，请登录到一个神话机器并运行以下命令之一：

/usr/class/cs143/bin/pa_fetch PA2<project_directory>

您指定的项目目录将在必要时创建，并将包含一些供您编辑的文件和一堆您不应该编辑的东西的符号链接。（事实上，如果您制作和修改这些文件的私有副本，您可能会发现无法完成作业。）请参阅 README 文件中的说明。 您需要修改的文件是：

- 酷

  此文件包含对 Cool 的解析器描述的开始。声明部分大部分是完整的，但是您需要为您介绍的新的非终端添加额外的类型声明。我们已经为您提供了终端的名称和类型声明。您可能还需要添加优先级声明。然而，规则部分相当不完整。 我们提供了一些规则的一些部分。 您不需要修改此代码以获得工作解决方案，但如果您愿意，欢迎您来。 然而，不要假设任何特定规则都是完整的。

- 好的

  这些文件测试语法的特性。 您应该添加测试，以确保 good.cl 练习语法的每一个合法构造，并且 bad.cl 在一个文件中尽可能多地练习各种类型的解析错误。 在这些文件中添加注释来解释您的测试。

- 自述文件

  此文件包含分配的详细说明以及一些有用的提示。您应该编辑此文件以包含您的 SUNETID（详见第 9 节）。

  虽然这些文件不完整，但解析器确实编译并运行。 要构建解析器，必须键入 make 解析器。

## 3 测试探测器

您将需要一个工作扫描仪来测试解析器。 您可以使用您自己的扫描仪或 coolc 扫描仪。 默认情况下，使

用 coolc 扫描器；若要更改此行为，请将 lexer 可执行文件（这是项目目录中的符号链接）替换为您自己的扫描器。不要自动假设扫描仪（无论你使用哪一个！）是无错误的-扫描器中的延迟错误可能会在解析器中引起神秘的问题。

您将使用 myparser 运行解析器，myparser 是一个 shell 脚本，它将解析器与扫描器“粘合”在一起。请注意，myparser 使用一个-p 标志来调试解析器；使用这个标志会导致大量关于解析器正在做什么的信息被打印在 stdout 上。野牛在 cool.out put 文件中生成 LALR（1）解析表的人可读转储。检查这个转储有时可能有助于调试解析器定义。

您应该在好的和坏的输入上测试这个编译器，看看是否一切正常。请记住，解析器中的错误可能在任何地方表现出来。

您的解析器将使用我们的词法分析器进行分级。因此，即使您使用自己的扫描仪完成大部分工作，您也应该在输入作业之前使用 coolc 扫描仪测试解析器。

## 4 解析器输出

您的语义操作应该构建 AST。AST 的根（而且只有根）应该是类型程序。对于成功解析的程序，解析器的输出是 AST 的列表。

对于有错误的程序，输出是解析器的错误消息。我们向您提供了一个错误报告例程，该例程以标准格式打印错误消息；请不要修改它。您不应该在语义操作中直接调用此例程；当检测到问题时，bison 会自动调用它。

对于跨越多行的构造，可以自由地将行号设置为构造的任何行。不要担心解析器报告的行是否与引用编译器完全匹配。此外，您的解析器只需要为包含在一个文件中的程序工作-不要担心编译多个文件。

## 5 错误处理

应该使用错误伪非终端在解析器中添加错误处理功能。错误的目的是允许解析器在一些预期的错误之后继续。它不是万能的，解析器可能会变得完全混乱。有关如何最好地使用错误，请参阅野牛文档。测试文件 bad.cl 应该有一些实例来说明解析器可以从中恢复的错误。要获得完整的信用，解析器至少应该在以下情况下恢复：

- 如果类定义中有错误，但类被正确地终止，下一个类在语法上是正确的，则解析器应该能够在下一个类定义中重新启动。
- 类似地，解析器应该从特性（继续到下一个特性）、let 绑定（继续到下一个变量）和{...}块。

不要过度关注解析器生成的错误消息中出现的行号。如果解析器工作正确，则行号通常是发生错误的行。对于跨越多行的错误构造，行号可能是构造的最后一行。

## 6 树包

在 Cool 文档的 Tour 部分中，对 Cool 抽象语法树包的 C++版本进行了广泛的讨论。您将需要大部分信息来编写工作解析器。

# 7 备注

您可以使用优先级声明，但仅用于表达式。 不要盲目地使用优先级声明（即不要通过添加优先级规则来响应语法中的移位减少冲突，直到它消失）。

Cool let construct 在语言中引入了歧义（如果你不相信的话，尝试构造一个示例）。 手册通过说 let 表达式尽可能向右扩展来解决歧义。根据语法的编写方式，这种歧义可能会在解析器中显示为涉及 let 的 productions 的移位减少冲突。 如果您遇到这样的冲突，您可能需要考虑通过使用 bison 特性来解决问题，该特性允许优先级与 production（而不仅仅是操作符）相关联。有关如何使用此功能的信息，请参阅野牛文档。

由于 mycoolc 编译器使用管道从一个阶段到下一个阶段进行通信，解析器产生的任何无关字符都可能导致错误；特别是，语义分析器可能无法解析解析器生成的 AST。 由于代码中留下的任何打印都会导致您丢失许多点，请确保在提交作业之前从代码中删除所有打印。

# 8 执行说明

您必须为具有属性的非末端和终端声明野牛的“类型”。 例如，在骨架 cool.y 中是声明：

类型<程序>程序

此声明表示非终端程序具有类型<程序>。 这里使用“type”一词是误导的；它的真正含义是，非终端程序的属性存储在 cool.y 中的联合声明的程序成员中，该声明具有类型程序。 通过指定类型

%型<member_name>XYZ...

您指示野牛，非末端（或终端）X、Y 和 Z 的属性具有适合于工会成员 member_name 的类型。

所有工会成员及其类型按设计都有类似的名称。 在上面的例子中，非终端程序与工会成员的名称是一致的。

您必须声明语法符号属性的正确类型；如果不这样做，实际上可以保证您的解析器无法工作。 您不需要为没有属性的语法符号声明类型。

- 在初始骨架文件上运行野牛将产生一些关于“无用的非终结者”和“无用规则”的警告。 这是因为一些非终点和规则永远不会被使用，但是当解析器完成时这些规则就会消失。
- 如果使用类型参数错误的树构造函数，则 g++类型检查器会抱怨。 如果忽略警告，当构造函数注意到程序被错误使用时，程序可能会崩溃。此外，如果你犯了类型错误，野牛可能会抱怨。听到任何警告。当 bison 或 g++发出警告信息时，如果您的程序崩溃，不要感到惊讶。

# 9 什么是自首

在提交之前，请确保您已经做了以下工作：

- 请确保您提交的最终版本没有任何调试打印语句。
- 请删除 README 中的说明部分。 本节是从开头开始，并运行到下面一行（包括这一行）

  --- 8<----------8<---------8<---------8<---- 切在这里-----8<---------8<---------8<---------8<----

  README 的这一部分只是为您提供关于需要编辑哪些文件的额外指导，您必须在实际提交作业之前删除它。
- 确保您已经编辑了 README，以便第一行以以下格式包含您的用户名：

  用户：<your_sunet_id>

  如果您在两个组中工作，应该有两个单独的用户行-如果两个 SUNETID 都包含在同一行中，则提交脚本将失败。 特别是，如果两个具有 SUNETID 的学生 abcdef 和 bcdefg 一起工作，他们的 README 应该以以下方式提到 SUNETID：

  用户：ABCDEF
  用户：bcdefg

  重要的是，以下内容将不起作用：

  用户：abcdef，bcdefg

  您的 SUNETID 是您用来登录神话机器的名称(而不是您的 8 位学生 ID 号)，可以使用 whoami 命令查询。
- 一旦 README 就绪，从项目目录中运行以下内容：

  提交

  这将打包项目目录中的大部分文件(它将忽略核心转储和 Emacs 备份文件等内容)，并将它们复制到提交文件夹中，同时附上时间戳。

  您提交的作业的最后一个版本将是分级的。 每次提交都覆盖上一次。 然而，我们保留保留旧的提交书的权利，以供在发生任何争端时参考。 请记住，您有三天的延迟时间用于整个季度；详情请参阅课程网站上的延迟政策页面。

  说服我们你理解材料的负担在你身上。 省略代码会对你的成绩产生负面影响，所以花点时间让你的代码可读性。
- 重要事项：我们将只接受通过提交脚本提交的作业。 请在截止日期前至少 24 小时测试-提交您的作业，以确保脚本为您工作，即使您没有完成。 提前检查问题是你的责任。