Trabalho Prático N.05 (Valor: 10 pontos)
Entrega: Conforme Tarefa no Canvas

1 Introdução

Neste trabalho, você implementará um gerador de código para a linguagem *Cool*. Quando realizado com sucesso, você terá um compilador completamente funcional para a linguagem *Cool*.

O gerador de código deve fazer uso da AST construída anteriormente, bem como da análise semântica que foi realizada já foi realizada. Seu gerador de código deve produzir código em linguagem de montagem ("assembly") do MIPS que implemente corretamente **qualquer** programa correto escrito em *Cool*. Não há nenhum tratamento/recuperação de erros na geração de código – todos os programas incorretos em *Cool* devem ter sido detectados pelos passos anterior do compilador.

Como ocorreu no trabalho de análise semâtica, este trabalho possui bastante espaço para decisões de projeto (que você deverá tomar). Sua implementação estará correta desde que o código que for gerado por ela funcione corretamente... Contudo a maneira de alcançar esse objetivo cabe inteiramente a você!!! Serão feitas algumas sugestões (como também discutidas em sala de aula) e convenções que se acredita tornar essa atividade mais fácil, porém você não é obrigado a segui-las.

Como em outros trabalhos, você deve explicar e justificar suas deciões de projeto no arquivo README. Este trabalho utiliza os mesmos códigos de suporte que o anterior, porém a quantidade total de código a ser produzido é bem maior... **Comeşe o mais breve possível!**

O fundamental para se obter um gerador de código correto é um entendimento tanto do comportamento de cada construção/instrução na linguagem *Cool* como a interface entre o código gerado e o sistema de suporte a execução (ambiente de execução – "*runtime system*").

O comportamento esperado de programas na linguagem *Cool* é definido por meio de semântica operacional apresentada na Seção 13 do *Manual de Referência de Cool*. Deve-se ressaltar que isso é apenas a especificação do significado das construções da linguagem e não como elas devem ser implementadas.

Já a interface entre o ambiente de execução ("*runtime system*") e o código gerado é descrita no material *Ambiente de Execução da Linguagem Cool* (*The Cool Runtime System*). Você deve consultar esse documento para uma discussão detalhada sobre o requisitos do ambiente de execução sobre o código gerado.

Há uma série de detalhes sobre este trabalho e sobre a geração de código nesse documentos, e você deve conhecê-los para conseguir implementar um gerador de código correto. **Leia toda essa documentação minunciosamente!**

Você pode trabalhar em grupo de até 05 pessoas para esta tarefa (grupos maiores não serão aceitos em nenhuma hipótese).

2 Arquivos e Diretórios

Para começar, crie um diretório onde você deseja fazer o desenvolvimento de seu trabalho (por exemplo, `PA5`) e execute um dos seguintes comandos *nesse diretório*. Para a versão em C++ do trabalho, você deve digitar:

```
cd PA5
make -f /var/tmp/cool/assignments/PA5/Makefile
```

Enquanto que para a versão em Java, digite:

```
cd PA5
make -f /var/tmp/cool/assignments/PA5J/Makefile
```

(observe o **"J"** no nome do caminho).

Este comando irá copiar um número de arquivos para o seu diretório. Alguns dos arquivos serão copiados somente para leitura (usando links simbólicos). Você não deve alterar esses arquivos. Na realidade, se você criar e modificar cópias particulares desses arquivos, poderá terminar por achar impossível concluir o trabalho. Veja instruções mais detalhadas no arquivo README.

A seguir, descreve-se os arquivos mais importantes para cada versão do projeto.

2.1 Versão em C++

Esta é uma lista dos arquivos que você pode querer modificar.

- cgen.cc
  Este é o arquivo que deverá conter a maior parte de sua implementação do gerador de código. O ponto de entrada para o seu gerador de código é o método **program_class::cgen(ostream&)**, que é chamado para a raiz da AST.

  Juntamente com as constantes usuais, foram fornecidas também funções para emissão de instruções do MIPS, um esqueleto para geração de código de strings, inteiros e booleanos; além de uma esqueleto para uma tabela de classes (**CgenClassTable**). Você pode ser utilizar o código fornecido ou substituí-lo se considerar mais adequado.

- cgen.h
  Este é arquivo de cabeçalho para o gerador de código. Você pode adicionar tudo aquilo que julgar necessário nesse arquivo. Ele fornece classes para implementação do grafo de herança. Você pode substitui-las ou modificá-las como desejar.

- emit.h
  Este arquivo contém várias macros para geração de código usadas na emissão de instruções MIPS entre outros. Você pode modificar esse arquivo a vontade.

- cool-tree.h
  Como anteriormente, este arquivo contém as declarações de classes para os nós da AST. Você pode adicionar declarações de atributos ou métodos às classes em cool-tree.h. A implementação dos métodos deverá ser adicionada ao arquivo cgen.cc.

- cgen_supp.cc
  Este arquivo contém o código de suporte geral para o gerador de código. Você encontrará aqui várias funções muito úteis. Você pode realizar adições a esse arquivo se desejar, **porém não altere nada que já esteja declarado aqui**.

- example.cl
  Este arquivo deve conter um programa teste que você irá implementar por sua conta. Procure testar a maior quantidade possível de construções de forma a garantir o correto funcionamento de seu gerador de código.

- README
  Este arquivo conterá anotações sobre a realização de sua tarefa. Para essa atribuição, é fundamental que você explique as decisões de design, como seu código está estruturado e os motivos que levam você a acreditar que o design é bom (ou seja, razões para sua implementação estar correta e robusta). Faz parte da tarefa explicar tais decisões por meio de texto e também comentar seu código. Arquivos README inadequados serão penalizados mais fortemente nesta atribuição, pois o README é a principal fonte de informações que se tem para entender seu código final.

2.2 Versão em Java

Esta é uma lista dos arquivos que você pode querer modificar.

- CgenClassTable.java e CgenNode.java
  Estes arquivos fornecem um implementação do grafo de herança para o gerador de código. Você deverá completar CgenClassTable para construir seu gerador. Você pode substitui-las ou modificá-las como desejar.

- StringSymbol.java, IntSymbol.java e BoolConst.java
  Estes arquivos fornecem suporte para constantes na linguagem *Cool*. Você deverá completar o método para geração de definição de constantes.

- cool-tree.java
  Este arquivo contém as definições para os nós da AST. Você deverá adicionar rotinas de geração de código (**code(PrintStream)**) para as expressões em *Cool* neste arquivo. O gerador de código é invocado por meio de uma chamada ao método **cgen(PrintStream)** da classe **program**. Você pode adicionar novos métodos, porém não modifique as declarações existentes.

- TreeConstants.java
  Como anteriormente, este arquivo define algumas constantes simbólicas úteis. Fique a vontade para adicionar as suas próprias, caso você julge necessário.

- CgenSupport.java
  Este arquivo contém o Este arquivo contém o código de suporte geral para o gerador de código. Você encontrará aqui várias funções muito úteis, incluindo algumas para emissão de instruções MIPS. Você pode realizar adições a esse arquivo se desejar, **porém não altere nada que já esteja declarado aqui**.

- example.cl
  Este arquivo deve conter um programa teste que você irá implementar por sua conta. Procure testar a maior quantidade possível de construções de forma a garantir o correto funcionamento de seu gerador de código.

- README
  Este arquivo conterá anotações sobre a realização de sua tarefa. Para essa atribuição, é fundamental que você explique as decisões de design, como seu código está estruturado e os motivos que levam você a acreditar que o design é bom (ou seja, razões para sua implementação estar correta e robusta). Faz parte da tarefa explicar tais decisões por meio de texto e também comentar seu código. Arquivos README inadequados serão penalizados mais fortemente nesta atribuição, pois o README é a principal fonte de informações que se tem para entender seu código final.

3 Projeto

Antes de prosseguir, sugere-se que você leia atentamente o material sobre *Ambiente de Execução da Linguagem Cool* (*The Cool Runtime System*) para se familiarizar com os requisitos sobre o seu gerador de código impostos pelo ambiente de execução.

Ao considerar seu projeto, em alto nível, seu gerador de código necessitará de realizar as seguintes tarefas:

1. Determinar e emitir/gerar código para constantes globais, tais como protótipos de objetos

2. Determinar e emitir/gerar cógigo para tabelas globais, tais como a **class_nameTab**, a **class_objTab** e as tabelas de despacho

3. Determinar e emitir/gerar código do método de inicialização para cada classe

4. Determinar e emitir/gerar código para cada definição de método.

Existem várias formas de se implementar um gerador de código. Uma estratégia razoável é realizar a geração de código em dois passos. O primeiro passo decide o "*layout*" dos objetos para cada classe, em particular o deslocamento ("offset") de cada atributo dentro da declaração de um objeto. Com base nesse informação, o segundo passo gerar (por meio de um caminhamento recursivo) o código de máquina de pilha adequado para cada expressão.

Existe vários pontos a se considerar durante o projeto de seu gerador de código:

- Seu gerador de código deve funcionar corretamente com o ambiente de execução ("runtime system") da linguagem *Cool*, conforme descrito no material sobre *Ambiente de Execução da Linguagem Cool* (*The Cool Runtime System*)
- Você deve ter uma ideia precisa e clara sobre a semântica de programas em *Cool*. A semântica de Cool é descrita informalmente na primeira parte do *Manual de Referência de Cool* e de forma bem precisa na Seção 13 no mesmo documento.
- Você deve compreender as instruções MIPS. Uma descrição do MIPS e suas instruções se encontra presente junto a documentação do simulador de MIPS denominado SPIM – ver *Manual do SPIM*.
- Você deve decidir quais invariantes seu código irá observar e esperar (isto é, que registradores serão salvos, quais poderão ser sobrescritos, etc).

Você **NÃO PRECISA** gerar o mesmo código produzido pelo compilador coolc. A implementação coolc inclui um alocador de registradores bem simples e outras pequenas modificações que não são necessárias para esse trabalho. O único requisito fundamental é que o código gerado ao final execute corretamente com o ambiente de execução ("runtime system") da linguagem *Cool*.

3.1 Verificação de Erro de Execução

Ao final do *Manual de Referência de Cool*, estão listados 06 (seis) erros que provocam o encerramento da execução de um programa.

Desses erros, o código gerado por você deve se preocupar em detectar os 03 (três) primeiros – despacho sobre void, case sobre void e inexistência de um opção válida em um sobre case – e, exibir uma mensagem de erro adequada antes de abortar.

Você poderá deixar o simular SPIM tratar da divisão por zero; enquanto que os outros dois últimos erros – intervalo inválido em substrings e estouro de memória – são de responsabilidade do ambiente de execução ("runtime system") implementado em `trap.handler`.

Veja a Figura 4 do material sobre *Ambiente de Execução da Linguagem Cool* (*The Cool Runtime System*) para uma listagem das funções que exibem mensagens de erro para você.

3.2 Coletor de Lixo

Sua implementação deve funcionar corretamente em conjunto com o coletor de lixo do ambiente de execução ("runtime system") de *Cool*. Os arquivos fornecidos contêm funções **code_select_gc** (C++) e **CgenClassTable.codeSelectGc** (Java) que geram código que *seta* opções para o coletor de lixo a partir de *flags* da linha de comando.

Os *flags* da linha de comando que afetam o coletor de lixo são -g, -t e -T. O coletor de lixo é desabilitado por *default*, o *flag* -g o habilita. Uma vez habilitado, o coletor de lixo não apenas recupera memória, mas também verifica se o valor "-1" separa todas as instâncias no *heap*; verificando dessa forma se o programa (ou mesmo, o coletor) não sobrescreveram acidentalmente o final de um objeto.

Os *flags* -t e -T são usados para testes adicionais. Com o *flag* -t, o coletor de lixo realiza recuperação de memória mais frequetemente (a cada alocação). O coletor não utiliza diretamente o *flag* -T; na implementação de compilador coolc, o *flag* -T produz código extraque realiza outras verificações durante a execução. Sinta-se livre para usar (ou não) o *flag* -T, caso deseje.

Para sua implementação, a forma mais simples é iniciar sem o uso do coletor (que já é o *default*). Quando você decidir utilizar o coletor, faça uma revisão cuidadosa da descrição da interface do coletor de lixo fornecida no material sobre *Ambiente de Execução da Linguagem Cool* (*The Cool Runtime System*). Assegurar de que seu gerador de código funciona corretamente com o coletor de lixo em **TODAS** as circunstâncias não é trivial.

4 Teste e Depuração

Você irá necessitar de analisadores léxico, sintático e semântico funcionais para testar seu gerador de código. Portanto, você pode utilizar os componentes que você desenvolveu nos trabalhos anteriores ou os componentes da implementação de coolc. Como padrão, os componentes de coolc serão usados. Para se modificar isso, substitua os executáveis lexer, parser e/ou semant (que são ligação simbólicas no seu diretório de projeto) pelas suas próprias implementações de analisadores (conforme desejar).

Mesmo que você utilize seus próprios componentes, é aconselhável testar seu gerador de código com os componentes da implementação de coolc, pois eles é que serão utilizados na avaliação de seu trabalho.

Você irá executar seu gerador de código usando o *script* mycoolc que junta o gerador produzido ao restante do compilador.

Observe que mycoolc recebe um *flag* -c para depuração do gerador de código; utilizando este *flag* simplesmente faz com que `cgen_debug` (uma variável global em C++ ou um atributo estático na classe `Flags` em Java) seja setada. A adição de código propriamente dito para produção de informação para depuração fica a seu encargo! Veja mais detalhes sobre isso no arquivo README.

4.1 Simuladores Spim e XSpim

Os programas spim e xspim são simuladores para a arquitetura MIPS com os quais você pode executar o código gerado.

O programa xspim funciona como o spim de modo oque ele permite você executar programas em linguagem de montagem para MIPS. Entretanto, ele possui várias facilidades que permitem você examinar o estado da máquina virtual, incluindo locais de memória, registradores, segmentos de dados e de código do programa. Você também consegui definir pontos de parada ("breakpoints") da execução e realizar execução passo a passo. Maiores informações podem ser obtidas na documentação fornecida para spim/xspim.

AVISO. A única dificuldade de depuração com spim está no fato dele ser na verdade um interpretador. Se suas definições de código ou dados fizerem referência a um rótulo indefinido, o erro somente aparecerá se algum código que for executado fizer referência explicita a tal rótulo. Além disso, um erro só será informado para rótulos indefinidos que apareçam na seção de código de seu programa. Se você possuir definições de dados constantes que façam referência a rótulos indefinidos, spim não irá informar nada a esse respeito! Ele simplesmente irá assumir o valor 0 (zero) para tais rótulos indefinidos!

5 O que entregar

Certifique-se de concluir os seguintes itens antes de enviar para evitar penalidades na avaliação de seu trabalho.

- Modificar o final do arquivo `README`. Parte dessa tarefa é preencher o `README` com anotações sobre o seu projeto.
- Inclua os casos de teste em example.cl.
- Assegure-se de que todo o seu código para o gerador de código está em nos seguintes arquivos:
  - cool-tree.h, cgen.h, cgen.cc, cgen_supp.cc e emit.h para a versão em C++, ou
  - cool-tree.java, CgenClassTable.java, CgenNode.java, CgenSupport.java, BoolConst.java, IntSymbol.java, StringSymbol.java, TreeConstants.java e arquivos adicionais .java (que você tenha acrescido) para a versão em Java.
- Compactar e codificar o subdiretório `PA5`, por exemplo da seguinte forma:

```
tar cvzf PA5.tar.gz PA5
uuencode PA5.tar.gz PA5.tar.gz > PA5.u
rm PA5.tar.gz
```

- Entregar o arquivo `PA5.u` via **Canvas**.