Trabalho Prático 4

1 Introdução

Neste trabalho, você irá implementar um gerador de código para `Cool`. Quando completado com sucesso, você terá um compilador de `Cool` completamente funcional! O gerador de código faz uso da AST construída no `TP3`. Seu gerador de código deve produzir código assembly MIPS que implemente *qualquer* programa `Cool` correto. Não existe recuperação de erro na fase de geração de código – qualquer programa `Cool` incorreto já deve ter sido detectado pelas fases de front-end do compilador.

Este trabalho tem bastante espaço para decisões de projeto. Seu programa está correto se o código que ele gera funciona corretamente; como você irá alcançar este objetivo fica a seu critério. Iremos sugerir algumas convenções que acreditamos que irá tornar sua vida mais fácil, mas você não precisa segui-las se não quiser. Como sempre, explique e justifique suas decisões no arquivo README.

Para a implementação do gerador de código, é crucial que você entenda tanto o comportamento esperado das construções da linguagem `Cool` quanto a interface entre o código gerado e o sistema de tempo de execução da linguagem. O comportamento esperado dos programas escritos em `Cool` é definido pela semântica oepracional descrita na Seção 13 do documento *The Cool Reference Manual*. Lembre-se que esta é somente uma especificação do significado das construções da linguagem – não uma maneira de implementá-las. A interface entre o sistema de tempo de execução e o código gerado é dado no documento *The Cool Runtime System*. Dê uma olhada nesse documento para uma discussão detalhada do sistema. Todos os arquivos de documentação podem ser encontrados na pasta "doc", distribuída através do arquivo "tp3-4.tar.gz". **O que deve ser entregue**: os arquivos modificados mais o arquivo README com a documentação do trabalho.

2 Arquivos e Diretórios

Para começar, extraia o arquivo "tp3-4.tar.gz" no local que preferir. Nesse arquivo, você encontrará uma pasta "cool". Após isso, crie o diretório em que você irá implementar o trabalho. O seguinte comando deve ser executado *dentro* do diretório criado:

```
> make -f caminho/para/cool/assignments/cgen/Makefile
```

Este comando irá copiar uma série de arquivos para o seu diretório. Alguns destes serão copiados apenas como leitura (*read-only*). Você **não deve** modificar tais arquivos. Dentre os arquivos copiados, você encontrará um arquivo README com algumas instruções. Os arquivos que você **deve** modificar são:

- cgen.cc: Este arquivo irá conter quase todo o código do seu gerador. O ponto de entrada para o seu gerador de código é o método **program_class::cgen(ostream&)**, que é chamado a partir da raíz da sua AST. Junto das constantes usuais, também fornecemos funções para emitir instruções MIPS, um esqueleto para codificar strings, inteiros e booleanos, e um esqueleto de uma tabela de classes (**CgenClassTable**).
- cgen.h: Este arquivo é o cabeçalho do gerador de código. Você pode adicionar o que quiser a ele. Ele fornece classes para implementação do grafo de heranças. Modifique-o como achar necessário.

- emit.h: Este arquivo contém várias macros de geração de código utilizadas para emissão de instrução MIPS, além de outras coisas. Você pode modificar este arquivo.
- cool-tree.h: Este arquivo contém as declarações de classes para os nós da AST. Você pode adicionar campos ou declarações de métodos a essas classes. A implementação desses métodos deve ser feita no arquivo cgen.cc.
- cgen_supp.cc: Este arquivo serve de auxílio ao gerador de código. Você vai encontrar uma série de funções utéis aqui. Adicione o que achar necessário, mas **não altere** nada que já esteja nele.
- example.cl: Este arquivo deve conter um programa de teste desenvolvido por você. Teste a maior quantidade de recursos do gerador de código que você conseguir.
- README: Este arquivo contém, inicialmente, uma lista com os arquivos copiados, além de algumas instruções para compilação e teste do projeto, bem como preparação do arquivo de submissão. Ao final do arquivo, você vai encontrar uma linha com os dizeres **"corte aqui"**. Abaixo desta linha você deve explicar qualquer decisão que tenha tomado quanto ao projeto, os casos de teste adicionados e por que você acredita que seu programa está correto. Antes de preparar o arquivo a ser submetido, certifique-se de que você apagou todo o conteúdo acima do "corte aqui" (incluindo a própria linha).

3 Projeto

Antes de continuar, sugerimos fortemente que você leia o documento *The Cool Runtime System* para se familiarizar. De maneira geral, seu gerador de código precisará realizar as seguintes tarefas:

1. Determinar e emitir código para constantes globais, tais como *prototype objects*;
2. Determinar e emitir código para tabelas globais, tais como **class_nameTab**, **class_objTab** e a tabela de *dispatch*;
3. Determinar e emitir código para os métodos de inicialização de cada classe; e
4. Determinar e emitir código para a definição de cada método

Existem diversas maneiras de escrever o gerador de código. Uma estratégia razoável é implementá-lo em duas fases. A primeira decide o layout de objeto para cada classe, particularmente o offset de cada atributo armazenado em um objeto. Usando esta informação, a segunda fase caminha, recursivamente, por cada *feature* e gera código de máquina de pilha (*stack machine code*) para cada expressão. Algumas coisas que você deve ter em mente ao projetar seu gerador de código:

- Seu gerador de código deve funcionar corretamente com o sistema de tempo de execução de `Cool`, que é descrito no manual *The Cool Runtime System*.
- Você deve ter um entendimento a cerca da semântica de tempo de execução de programas escritos em `Cool`. A semântica está descrita informalmente no primeiro parágrafo do documento *The Cool Reference Manual* e mais precisamente na Seção 13 (do mesmo manual).
- Você deve compreender o conjunto de instruções da arquitetura MIPS. Uma visão geral das operações MIPS é dado na documentação do spim (também na pasta "doc").

- Você deve decidir quais invariantes seu gerador de código vai observar e esperar (e.g. quais registradores devem ser salvos, quais podem ser sobrescritos, etc).

Você não precisa gerar o mesmo código que coolc; coolc inclui um alocador de registradores simples e algumas outras alterações que não são necessárias para este trabalho. O único requisito é gerar código que execute corretamente no ambiente de execução.

3.1 Tratamento de Erro em Tempo de Execução

O final do manual de `Cool` lista seis erros que fazem o programa abortar. Destes, o código gerado por você deve capturar os três primeiros (conferir no manual) e imprimir uma mensagem de erro adequada antes de abortar. Veja a Figura 4 do documento *The Cool Runtime System* para uma lista de funções que imprimem as mensagens de erro adequadas.

3.2 Coletor de Lixo

Para receber nota máxima neste trabalho, seu gerador de código deve funcionar corretamente com o coletor de lixo geracional (GC, do inglês *Garbage Collector*) de `Cool`. O template que você recebeu contém uma função **code_select_gc** que define opções do GC através de opções passadas por linha de comando. As opções que afetam o funcionamento do GC são `-g`, `-t` e `-T`. O GC está desabilitado por padrão; a opção `-g` o habilita. Quando habilitado, o coletor não apenas libera memória, mas também verifica se "-1" separa todos os objetos no heap, verificando dessa forma se o programa (ou o próprio coletor) não sobrescreveu acidentalmente o fim de um objeto. As opções `-t` e `-T` são utilizadas para testes adicionais. Com a opção `-t`, o GC irá realizar a coleta frequentemente (a cada alocação). O GC não utiliza diretamente `-T`; em `coolc`, a opção `-T` faz com que código extra seja gerado, código este que realiza mais validações. Você está livre para utilizar (ou não) a opção `-T` como desejar. Para a sua implementação, a maneira mais simples de começar é não utilizar o GC (é o comportamento padrão). Quando decidir utilizar o coletor, lembre-se de revisar cuidadosamente a interface do GC descrita em *The Cool Runtime System*. Garantir que seu gerador funciona corretamente com o GC em todos os casos não é trivial.

4 Testando e Debugando

Você utilizará implementações padrão do analisador léxico, sintático e semântico. No caso do sintático, se assim desejar (e tiver certeza que seu parser está correto) você pode utilizar o parser que você desenvolveu no trabalho anterior. Para isso, basta substituir o executável parser que foi copiado pelo executável correspondente a sua implementação.

Você executará o gerador de código através do script mycoolc, que une todas as fases de compilação. Note que mycoolc aceita uma opção `-c` para debug do gerador de código; o que esta opção faz é apenas definir o valor da variável `cgen_debug`. Adicionar código que produz, de fato, informações úteis de debug fica a sua escolha. Verifique o arquivo README para mais detalhes.

4.1 Spim

Para testar os programas gerados, você irá uilizar um emulador para MIPS chamado spim (você terá que instalá-lo). Uma vez que o spim esteja instalado, basta utilizar o script spim que foi copiado para a pasta em que está trabalhando:

```
> ./spim -file example.s
```

Esse script irá chamar o spim, passando como opção -exception_file o arquivo caminho/para/cool/lib/trap.handler, que contém o código relacionado a inicialização, tratamento de exceções, etc. Alternativamente, você pode instalar o xspim (ou QtSpim, a versão mais nova do emulador), que é semelhante ao spim, mas com recursos adicionais que te permitem examinar o estado da máquina virtual, incluindo memória, registradores, segemento de dados e segmento de código do programa. A documentação dos programa spim/xspim estão incluídas na pasta "doc". Caso deseje utilizar o xspim (ou QtSpim), lembre-se de utilizar o arquivo trap.handler; caso contrário, ele irá utilizar o arquivo padrão e você terá problemas.