Projeto de Compiladores
Parte 4

A quarta parte do projeto consistirá na geração de código no formato da representação intermediária *LLVM* (*LLVM IR)*.
Será fornecido um arquivo *Grammar.g4* com a implementação da gramática e um arquivo *GrammarCheckerVisitor.py* com a implementação do *visitor* da terceira parte do projeto para quem eventualmente não conseguiu completá-la.
Vocês devem implementar o *visitor* de forma a gerar códigos *LLVM IR* aceitos e executáveis pelo compilador *Clang*. Os seguintes requisitos devem ser cumpridos:

Requisitos:

- O visitor implementado deve gerar como saída um arquivo com nome *output.ll* contendo o código *LLVM IR* equivalente ao programa passado como entrada.
- O código gerado deve ser válido e aceito pelo compilador *Clang*.
- Vocês devem utilizar a simplificação de expressões constantes desenvolvidas no projeto 3 para simplificar o código gerado.
  - Expressões com valores constantes, incluindo variáveis constantes, devem ser colocadas no código *LLVM IR* como valores constantes.
- Não será obrigatória a implementação de *branches* no código. Ou seja, não precisam implementar a geração de código para *if*, *else* ou *for*.

Para fins de teste, vocês podem gerar um código *LLVM IR* utilizando o seguinte comando do *Clang*:

clang -S -emit-llvm -O0 inputFile.c

Onde é possível alterar a flag de otimização -O0 para -O1, -O2 ou -O3, sendo -O0 sem otimização e -O3 todas as otimizações disponíveis.

Os códigos gerados pelo *Clang* possuem diversas partes que não são necessárias para que o código esteja correto, como as linhas iniciadas com ‘;’, que são comentários, entre outros atributos. Caso queira ver uma versão simplificada do código, contendo apenas o básico necessário, analise os arquivos dentro do diretório *outputs*.

Caso queiram compilar e gerar um executável a partir de um código *LLVM IR*, vocês podem executar o seguinte comando:

clang inputFile.ll -o outputFile

e em seguida executar o executável gerado.

DICAS:

- Para quem vai rodar no Windows, leia o Readme disponibilizado no diretório.
- Os arquivos de saída dão uma boa noção de como escrever um código *LLVM IR*. Consulte-os enquanto estiver implementando o visitor.
- Você também pode gerar códigos *LLVM IR* a partir de qualquer arquivo C utilizando o *Clang*.
- Consulte a documentação do *LLVM IR* para entender cada operação.
- O download do LLVM e do Clang podem ser feitos neste link.
- Constantes do tipo *float* devem ser escritas utilizando seu valor hexadecimal (o valor hexadecimal equivalente aos bits da representação de um *floating point*). Para isto, utilize a função *float_to_hex* já disponível em *GrammarCheckerVisitor.py*. Explicação para esta necessidade neste link.