prof. Éfren L. Souza

UFOPA – Universidade Federal do Oeste do Pará
IEG – Instituto de Engenharia e Geociências
PC – Programa de Computação
Disciplina – Compiladores

Trabalho de Compiladores

Este trabalho consiste em adicionar novos recursos à linguagem DL, usando como base o compilador implementado em sala de aula (`dl_short_4`). Todos os recursos devem passar por todas as etapas de análise e gerar o código LLVM-IR adequado.

Algumas dicas importantes na implementação deste trabalho são: (i) utilize o compilador `Clang` para ajudar a entender o processo de geração de código intermediário; (ii) pesquise pela gramática (BNF) de linguagens populares, como C, C++, Java e Python, para ajudar na construção da sintaxe; e (iii) consulte os códigos do Livro do Dragão (Apêndice A), apesar dele construir uma linguagem diferente, os códigos do livro podem dar uma boa ideia de como construir a DL.

A gramática da linguagem DL está descrita no arquivo `README` na raiz do projeto. Adicionar novos recursos à linguagem pode exigir modificações na gramática, logo qualquer alteração na gramática DEVE ser registrada no arquivo `README`.

A equipe deve entregar apenas o projeto do compilador com os recursos solicitados. Cada equipe ficará responsável por acrescentar alguns recursos à linguagem. As especificações desses recursos e as esquipes responsáveis são:

1. Equipe 1
    a. **Literais inteiros hexadecimal e binário**: Muitas linguagens permitem usar literais em outras bases numéricas. Faça a linguagem DL aceitar literais inteiros do tipo hexadecimal e binário, utilize a mesma sintaxe do C/C++.
    b. **Operadores de divisão (/) e resto da divisão (`%`)**: Adicione os operadores de divisão e resto, com a mesma prioridade da multiplicação. Observe que o lexema do operador de divisão é o mesmo que inicia os comentários.
    c. **Comando ELSE**: O comando **`se`** da linguagem DL, por enquanto, não aceita o seu par **`senao`**. Adicione esse comando usando a seguinte regra de produção **`se`** `(EXPR) STMT` **`senao`** `STMT`.
2. Equipe 2
    a. **Operador de potenciação**: Algumas linguagens como Python e Matlab têm um operador de potenciação associativo à direita. Adicione esse

operador à linguagem DL usando a mesma sintaxe e funcionamento do Python. Um detalhe importante: se você usar só operandos inteiros nessa operação, o resultado deve ser inteiro, caso tenha algum operando real, o resultado deve ser real.

   b. **Comando Leia**: Comando de entrada e saída de dados são essenciais para qualquer linguagem. A linguagem DL já possui o comando de saída de dados **`escreva`**`(ID)`, então adicione o comando de entrada de dados **`leia`**`(ID)`, que é capaz de ler valores para inteiros e reais.
   c. **Comando WHILE**: Adicione o comando de repetição **`enquanto`**`(EXPR) STMT`, seu funcionamento é igual ao do C/C++.

3. Equipe 3
   a. **Operadores de igualdade e relacionais**: Adicionar o operador de igualdade (==) diferença (`!=`) e maior ou igual (`>=`). Observe as precedências desses operadores.
   b. **Operadores Lógico AND e NOT**: Adicionar os operadores lógicos AND (`&`) e NOT (!), lembrando que o operador NOT é unário. Consulte a documentação do C/C++ e aplique as mesmas precedências dessa linguagem para a linguagem DL. Esses operadores devem possibilitar tanto a computação de valores lógicos quanto o controle de fluxo.
   c. **Atribuir expressão real para uma variável inteira**: As regras semânticas da DL permitem que uma expressão inteira seja atribuída a uma variável real, mas não o contrário. Modifique as regras para que o segundo caso também seja aceito.
4. Equipe 4
   a. **Operadores unários de soma e subtração**: adicione as operações unárias de soma e subtração, gerando o código intermediário apropriado.
   b. **Inc/Dec pré-fixado**: Adicione as operações de incremento (++) e decremento (--) pré-fixados. Esses operadores só são aplicáveis a variáveis numéricas. Consulte a documentação e a BNF do C/C++ e construa o mesmo funcionamento e precedência na DL.
   c. **Inc/Dec pós-fixado**: Adicione as operações de incremento (++) e decremento (--) pós-fixados. Esses operadores só são aplicáveis a variáveis numéricas. Consulte a documentação e a BNF do C/C++ e construa o mesmo funcionamento e precedência na DL.
5. Equipe 5
   a. **Literal inteiro romano**: Adicione literais inteiros usando algarismos romanos. Esse deve ter o prefixo `0r` seguido de um ou mais algarismos romanos (i, v, x, l, c, d, m) maiúsculos ou minúsculos. Lembre-se que há regras na ordem de cada algarismo, portanto há sequências de algarismos romanos que não formam um número romano válido, nesse caso um erro léxico deve ocorrer. O maior número a ser representado por esse tipo de literal deve ser pelo menos 4999.

   b. **Tabela de Símbolos por escopo**: Adicione o escopo de variáveis à linguagem DL, i.e., cada bloco de comandos tem seu próprio ambiente. Consulte o capítulo 2.7.1 e o Apêndice A do Livro do Dragão e use a mesma regra de escopo (regra de aninhamento mais interno para blocos) para a linguagem DL.
   c. **Comando REPEAT**: O comando repita é similar aos comandos `REPEAT-UNTIL` do Pascal e `DO-WHILE` do C/C++, ou seja, eles primeiro executam o corpo do laço, para depois verificar a condição de repetição. Adicione esse comando seguindo o mesmo funcionamento do Pascal. Use a seguinte produção **`repita`** `STMTS` **`ate`** `(EXPR)`. Observe que o corpo desse comando tem um bloco próprio (não utiliza `BLOCK`), logo ele possui um ambiente próprio.
6. Equipe 6
   a. **Comentários**: Fazer com que o analisador léxico ignore comentários de uma linha e comentários de múltiplas linhas no mesmo estilo do C/C++. Garanta que os comentários de múltiplas linhas estejam fechados, caso contrário deve ocorrer um erro léxico.
   b. **Literal real em notação científica**: Os literais reais também podem ser representados em notação científica. Faça a linguagem DL aceitar essa representação da mesma forma que a linguagem C/C++. Recomento fortemente que o diagrama de estados que reconhece esse lexema seja desenhado antes de partir para o código.
   c. **Operadores de atribuição com aritmética**: Operadores de atribuição `+=`, `-=` e `*=`, funcionando da mesma forma que na linguagem C/C++.

Os critérios de avaliação do trabalho são:

a) Ajuste adequado da gramática (2 pontos);
b) Fidelidade às etapas do processo de compilação (3 pontos);
c) Funcionamento dos recursos adicionados (5 pontos).